# TrueFi v2 Smart Contracts Audit

## Intro

The TrustToken team is building an uncollateralized lending platform. Version 1 was launched in November 2020; they asked me to audit version 2.0 in preparation for its scheduled February 2021 launch. 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/truefi
- TrueRatingAgencyV2.sol
- LoanToken.sol
- Liquidator.sol
- TruPriceUniswapOracle.sol

The version of the code I reviewed was published with the commit b8189766c4bf22660754c13dd7c178766b333fd0.

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

- One critical severity issue was found: a denial-of-service vulnerability in the new ERC20 token stkTRU which could cause the token to become unusable. This issue must be fixed before launch.
- Two medium and one low severity issues were found, which in edge cases could allow an incorrect number of staked TRU to slashed during a loan liquidation, TRU to be staked and then unstaked without waiting for a whole cooldown period, and voting power to be retroactively diluted after a loan liquidation.

# Issues found


Denial-of-Service vulnerability in payFee (CRITICAL)

Uniswap TRU price can be manipulated (MEDIUM)

Staking in unstake period should reset cooldown (MEDIUM)

Retroactive vote dilution (LOW)

Notes

- Assumption that every token is either TRU or tfUSD
- Division by totalSupply when totalSupply could be zero
- Differing units should be better documented
- Cooldown event is not emitted on cooldown reset
- Increasing cooldown times also increases current cooldowns
- Increasing cooldown times may create unexpected short cooldowns
- Dead code in TrueRatingAgencyV2

# Denial-of-Service vulnerability in payFee

## Severity: CRITICAL (Impact: HIGH, Likelihood: HIGH)

Problem

Any user can call **payFee** in StkTruToken.sol, and each invocation increases the effective length of **scheduledRewards**, which increases the work that must be done by **distributeScheduledRewards**.

Consequences

If an attacker (or sufficient legitimate usage) uses this to grow **scheduledRewards** large enough, then later invocations of **distributeScheduledRewards** will fail since the while loop on line 372 will exceed the block gas limit. This would cause all functionality relying on **distributeScheduledRewards** (which is almost everything) to become permanently unusable.

Recommendations

I haven't thought of a full solution yet, but mitigation possibilities include:

- Reduce Likelihood: Since the only *intended* caller of **payFee** seems to be TrueLender.sol, you could restrict **payFee** to only be callable by that contract. This would prevent this DoS from being triggered directly by random attackers, but would *not* help against increased legitimate usage.
- Reduce Impact via fallback functionality: You could restrict the maximum effective length of **scheduledRewards**, by checking if it is about to get too large and then either reverting the transaction or releasing pending rewards early. Or you could bound the loop so that it can't fail (but also might not release all pending rewards). (Neither is ideal but both are better than the complete lockout you currently face.)
- Reduce Impact via pressure release valve: You could provide an external function which works just like **distributeScheduledRewards** except the loop is bounded so it can't fail. This would allow anyone to release the pressure on **scheduledRewards** if it is growing too large.

# Uniswap TRU price can be manipulated

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

Problem

Liquidator.sol uses the current Uniswap price of TRU (accessed through TruPriceUniswapOracle.sol) to determine how much staked TRU to slash when a loan defaults. However, this price can be significantly manipulated by trading with Uniswap.

Consequences

An attacker could do the following in a single transaction:

1. (Optional: take out a flash-loan to fund the next step)
2. Trade a large amount of TUSD for TRU on Uniswap, driving up the price of TRU
3. Liquidate a loan.
4. Reverse the Uniswap trade (thereby recouping all slippage, though not trading fees)
5. Repay the flash loan if taken out in step 1

Since TRU is artificially highly-priced during the liquidation in step 3, an inappropriately small amount of TRU is slashed in the liquidation. Thus the purpose of this attack might be for the attacker to protect some of their staked TRU from being slashed (at the expense of tfUSD holders). Similarly, the attacker could instead temporarily crash the price of TRU in step 2 to cause an inappropriately *large* amount of TRU to be slashed, in order to increase the value of their tfUSD holdings (at the expense of TRU stakers).

Based on the Uniswap liquidity at time of writing (16 Feb 2021), performing this attack using 1 million TUSD in step 2 would cause a price impact of ~64% (and 10 million for ~94%). Temporary price manipulations will be easier in the future if Uniswap liquidity decreases, which is likely based on my understanding that the community plans to shift incentives away from Uniswap towards Sushiswap.

Recommendations

Use the average from multiple oracles and/or use an average price over time to make the price harder to manipulate.

# Staking in unstake period should reset cooldown

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

Problem
In StkTruToken.sol, **stake** does not reset the cooldown timer if called during the unstake period.

Consequences
A user could call **cooldown** with nothing staked, and then only stake at the beginning of the 2-day unstake period. In this way they could earn full staking rewards for those 2 days without being subject to normal token lockup. In fact, a user could put 8 addresses on a rotating cooldown schedule, such that by moving their stake between the addresses, they could earn staking rewards continuously while never having their tokens locked.

Recommendations
Staking during a cooldown already resets the cooldown; staking during unstake period should also reset the cooldown.

# Retroactive vote dilution

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

Problem

**getPriorVotes** in StkTruToken.sol computes an account's past voting power using the *current* **stakeSupply** and **totalSupply** rather than what those values were in the past.

Consequences

The rest of TrueFi governance is a snapshot-style system: your voting power for a given proposal depends only on your token holdings and delegates at the time the proposal was made, regardless of what trades, burns, etc may have happened afterward. However, this issue with **getPriorVotes** creates a seemingly-unintended exception: when TRU is liquidated, the voting power of stkTRU holders goes down not only on future proposals (as intended) but also on past ones.

Recommendations

If this is actually an intended exception to snapshot-style voting, document it. Otherwise, you could extend the snapshot system to also record the TRU-per-stkTRU over time, and then reference that when computing prior voting power.

# Notes

## Issues with a very low impact

### Assumption that every token is either TRU or tfUSD

In StkTruToken.sol, there are five methods with an argument **token** of type **IERC20**. There is an undocumented assumption that the given token will always be one of TRU and tfUSD - for example, **rewardBalance(token)** will return the tfUSD balance if **token** is *anything* other than TRU. In an abundance of caution I would add explicit checks for this in the one relevant external function (**claimable**). For the remaining functions, I don't see security implications so checks on the input might not be worth the gas, but documentation in comments would help future developers and readers.

### Division by totalSupply when totalSupply could be zero

In StkTruToken.sol lines 281, 300, and 311, the **div(totalSupply)** will fail if **totalSupply** is zero, even though all these methods do have sane output values they might still be expected to produce in that case (e.g. **getCurrentVotes** could output zero instead of crashing). However **totalSupply** will likely *never* be zero (after a short start-up phase), so this is of minimal importance.

### Differing units should be better documented

In StkTruToken.sol, some variables store actual numbers of tokens, while others store numbers of tokens multiplied by **PRECISION** (10^30). For example, if 6 TRU (ignoring **decimals**) have been distributed from the farm and all claimed, then **farmRewards[tru].totalClaimedRewards** will be 6 but the similar-sounding **farmRewards[tru].totalFarmRewards** will be 6*10^30. To prevent confusion, this difference should be documented in comments and possibly also in the variable names (e.g. by inserting the prefix "quecto" (10^-30) into **totalFarmRewards** et al).

### Cooldown event is not emitted on cooldown reset

In StkTruToken.sol, staking new tokens can restart a cooldown, but no event is emitted when this happens. You may want to emit a new **Cooldown** event containing the updated cooldown end time. (On the other hand, staking probably correlates with a desire to not unstake, plus there are other ways for a user to check the updated cooldown end time, so maybe emitting a **Cooldown** event along with the **Stake** event isn't actually useful to anyone.)

### Increasing cooldown times also increases current cooldowns

In StkTruToken.sol, increasing cooldowns through **setCooldownTime** also increases cooldowns currently in progress. For example, a user starts a (14-day) cooldown on day 10.

Then on day 20 governance calls **setCooldownTime(90 days)**, and now the user has to wait until day 100 to unstake. Users may expect the new cooldown rule to only apply to cooldowns begun after the rule change, which could be accomplished by storing end times instead of start times in **cooldowns[]**.

## Increasing cooldown times may create unexpected short cooldowns

In StkTruToken.sol, increasing cooldowns through **setCooldownTime** can occasionally *decrease* cooldowns for existing users. For example, a user starts a (14-day) cooldown on day 10, unstakes on day 24, and restakes on day 100. That day, governance calls **setCooldownTime(90 days)**, and suddenly the user is able to unstake immediately (thanks to the unrelated cooldown begun on day 10) even though they just staked.

## Dead code in TrueRatingAgencyV2

In TrueRatingAgencyV2.sol, some code and comments are relics of the old v1 staking system:

- The condition on line 430 can no longer ever be true, so the whole if-block on lines 430-433 is dead code and should be deleted.
- The comment on line 377 is wrong and should be deleted (claiming rewards is now fully independent of the defunct notion of unstaking from loans)

*Benjamin Cosman*