Smart Contract Security Audit Report

Secured Finance Stablecoin

1. Contents

2. General Information

This report contains information about the results of the security audit of the Secured Finance (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 19/02/2025 to 05/03/2025.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: stablecoin-contracts. Initial review was done for the commit f92b9a. Re-test was done for commit 0a7db6.

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered a three medium security issue. The other suggestions included fixing the low-risk issues and some best practices. The Secured Finance team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of March 10, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Tellor oracle is not protected from dispute attack	contracts/Dependencies/TellorCaller.sol	Medium	Fixed
Missing storage gaps in intermediate contracts	contracts/Dependencies/TroveBase.sol contracts/LPRewards/Unipool.sol	Medium	Fixed
Pyth Oracle Price Manipulation		Medium	Acknowledged
Missing call to _disableInitializers	contracts/*	Low	Fixed
Lower dispute buffer	contracts/Dependencies/TellorCaller.sol	Low	Fixed
Confidence interval is ignored	contracts/Dependencies/PythCaller.sol	Low	Acknowledged
Wrong comments	contracts/ProtocolToken/ProtocolToken.sol contracts/PriceFeed.sol	Low	Fixed
Bad naming in variable declaration	contracts/Dependencies/TellorCaller.sol contracts/Proxy/ProtocolStakingScript.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Tellor oracle is not protected from dispute attack

Risk Level: Medium

Status: Fixed in the commit 28b35e.

Contracts:

- contracts/Dependencies/TellorCaller.sol

Location: Function: `getTellorCurrentValue()`.

Description:

Tellor oracle allows any TLR staker to submit a price, and disputes can be raised by paying a small TRB fee. When a price report is disputed, it is removed from the available data, and TRB holders vote on its validity.

In the current implementation, a malicious user can dispute valid price reports to force the contract to use older, potentially more favorable values. This could enable them to manipulate collateral purchases at advantageous prices.

Remediation:

Cache the most recent valid price and timestamp to ensure that a single dispute cannot force the use of outdated data.

References:

- https://github.com/tellor-io/tellor-caller-liquity/tree/main

5.2. Missing storage gaps in intermediate contracts

Risk Level: Medium

Status: Fixed in the commit e58732.

Contracts:

- contracts/Dependencies/TroveBase.sol
- contracts/LPRewards/Unipool.sol

Description:

In Solidity, storage gaps are crucial in upgradeable contracts to prevent storage layout conflicts when adding new variables in future upgrades. However, the intermediate contracts: `LPTokenWrapper` in `Unipool` and `TroveBase` in `TroveManager` do not include storage gaps.

This creates a risk where future changes could overwrite storage variables in an unintended way, potentially breaking contract functionality or corrupting stored data.

```
contract Unipool is LPTokenWrapper, OwnableUpgradeable, CheckContract,
IUnipool {

contract TroveManager is TroveBase, OwnableUpgradeable,
ReentrancyGuardUpgradeable, ITroveManager {
```

Remediation:

Consider using gaps at the end of the described contracts to prevent the possibility of storage overriding.

5.3. Pyth Oracle Price Manipulation

Risk Level: Medium

Status: Not needed to fix it because:

- We have already an off-chain monitoring module that automatically triggers price updates.

- We expect more DeFi projects become active on FVM. In that situation, Pyth price feeds are updated more frequently by many projects.

- We might change the price oracle if other price oracles support FVM in the future. For that, we don’t want to create Pyth-specific logic.

Description:

The contract is vulnerable to price manipulation due to insufficient validation of price update age in the Pyth oracle implementation. The oracle operates on a pull model where users must manually submit price updates.

The vulnerability stems from several factors:

- The oracle prices are updated by users on average once every few hours

- The contract lacks checks for the age of the submitted price data beyond the 1-day timeout
- The `updateFeeds()` function allows users to submit any valid historical price data that is newer than the last updated price, even if it's not the most recent

An attacker could exploit this vulnerability through the following steps:

1. Wait for significant price volatility (>10% change) in FIL price
2. Use a flash loan in FIL to execute the attack, allowing the attacker to maximize their position size and potential profits
3. Submit outdated (but still valid) price data showing the highest price since the last oracle update
4. Mint USDFC using FIL as collateral at the manipulated (higher) price
5. Exchange the minted USDFC back to FIL in liquidity pools at the current (lower) market price
6. The attacker ends up with more FIL than they initially deposited, as they borrowed USDFC at an artificially high FIL price and repaid it at the true lower price
7. Once the actual (lower) market price is reflected in the oracle, the position becomes undercollateralized. Given the system's 110% minimum collateralization requirement, the 10% price decline pushes the collateral ratio below the 100%

Remediation:

8. Consider implementing strict freshness checks for price updates and require users to update the price when opening new positions
9. Consider increasing collateral requirement to 120% to account for FIL's potential volatility
10. Consider implementing an off-chain monitoring module that automatically triggers price updates when price movements exceed 5% to maintain price accuracy and prevent manipulation

5.4. Missing call to _disableInitializers

Risk Level: Low

Status: Fixed in the commit 68c836.

Contracts:

- contracts/*

Location: Function: `constructor()`.

Description:

The constructor of the contracts does not call the `_disableInitializers()` function. This is a recommended practice to prevent the contract from being initialized accidentally.

Remediation:

Add a call to `_disableInitializers()` in the constructor.

References:

- https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable

5.5. Lower dispute buffer

Risk Level: Low

Status: Fixed in the commit 88714e.

Contracts:

- contracts/Dependencies/TellorCaller.sol

Description:

When Pyth oracle is unable to work protocol switches to Tellor oracle. The current dispute buffer is 20 minutes which is too much for such volatile asses as FIL. According to the Liquity research of ETH volatility, 15 minutes is the optimal value.

Remediation:

Consider lowering the buffer to 15 minutes

References:

- https://www.liquity.org/blog/tellor-issue-and-fix

5.6. Confidence interval is ignored

Risk Level: Low

Status: Not needed to fix it because we don't want our liquidation feature stop under highly volatile situations.To reduce this risk, we let off-chain price monitoring module work more frequently under highly volatile situations to fetch stable prices ASAP.

Contracts:

- contracts/Dependencies/PythCaller.sol

Location: Function: `getRoundData()` `latestRoundData()`.

Description:

The prices fetched by the Pyth network come with a degree of uncertainty which is expressed as a confidence interval around the given price values. Considering a provided price p, its confidence interval σ is roughly the standard deviation of the price's probability distribution. The official documentation of the Pyth Price Feeds recommends some ways in which this confidence interval can be utilized for enhanced security. For example, the protocol can compute the value σ / p to decide the level of the price's uncertainty and disallow user interaction with the system in case this value exceeds some threshold.

Currently, the protocol completely ignores the confidence interval provided by the price feed.

Remediation:

Consider utilizing the confidence interval provided by the Pyth price feed as recommended in the official documentation. This would help mitigate the possibility of users taking advantage of invalid prices.

```
        if (conf > 0 && (price / int64(conf) < MIN_CONFIDENCE_RATIO)) {
            revert("Untrusted price");
        }
```

5.7. Wrong comments

Risk Level: Low

Status: Fixed in the commit ac3404.

Contracts:

- contracts/ProtocolToken/ProtocolToken.sol
- contracts/PriceFeed.sol

Description:

The contract contains **outdated or incorrect comments**, leading to **misleading documentation** about token allocation and price feeds. These inconsistencies make it harder to understand the correct logic and might result in incorrect assumptions.

In ProtocolToken.sol, the comments incorrectly state the allocation of tokens:

```
contracts/ProtocolToken/ProtocolToken.sol:
  21  * transfer() and transferFrom() calls. The purpose is to protect users
from losing tokens by mistakenly sending ProtocolToken directly to a core
contract,
  22  * when they should rather call the right function.
  23  *
  24  * 2) sendToProtocolTokenStaking(): callable only by core contracts,
which move ProtocolToken tokens from user -> ProtocolTokenStaking contract.
  25  *
  26: * 3) Supply hard-capped at 100 million
  27: *
  28  * 4) CommunityIssuance addresses are set at deployment
  29  *
  30  * 5) The bug bounties / hackathons allocation of 2 million tokens is
minted at deployment to an EOA
  31
  32  * 6) 32 million tokens are minted at deployment to the CommunityIssuance
contract
```

However, the actual allocation from deployment script shows a different breakdown:

```
deployments/inputs/mainnet.js:
  31: const allocationAmounts = {
  32:   ADMIN: dec(6_700_000, 18),
  33:   UNIPOOL: dec(100_000, 18),
  34:   COMMUNITY_ISSUANCE: dec(3_200_000, 18),
  35: };
```

The price feed contract still references **Chainlink**, even though **Pyth** is being used instead**:**

```
contracts/PriceFeed.sol:
   27:     AggregatorV3Interface public priceAggregator; // Mainnet Chainlink
aggregator
```

Remediation:

Consider updating all token allocation comments to match the actual deployment values and Replacing Chainlink references with Pyth in relevant contract comments.

5.8. Bad naming in variable declaration

Risk Level: Info

Status: Fixed in the 28b35e and 0a7db6 commits.

Contracts:

- contracts/Dependencies/TellorCaller.sol
- contracts/Proxy/ProtocolStakingScript.sol

Description:

Certain variables in the contract have unclear naming making the code harder to read and maintain.

```
contracts/Proxy/ProtocolStakingScript.sol:
  9:     IProtocolTokenStaking immutable protocolStakingStaking;
```

The query is for FIL, not BTC:

```
contracts/Dependencies/TellorCaller.sol:
  21:     bytes32 public immutable btcQueryId;
```

Remediation:

Consider modifying variables names.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.