Sturdy (fixes)

Security Review

Cantina Managed review by:

Desmond Ho, Lead Security Researcher

Xmxanuel, Security Researcher

October 24, 2023

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Directly* exploitable security vulnerabilities that need to be fixed.
High	Security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All high issues should be addressed.
Medium	Objective in nature but are not security vulnerabilities. Should be addressed unless there is a clear reason not to.
Low	Subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of low severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or high. Critical findings should be directly vulnerable and have a high likelihood of being exploited. High findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

2 Security Review Summary

Sturdy enables anyone to create a liquid money market for any token. Sturdy uses a novel two-tier architecture to isolate risk between assets while avoiding liquidity fragmentation. The base layer consists of risk-isolated pools; aggregation built on top enables lenders to select which collateral assets can be used as collateral for their deposits.

Contracts in scope include:

```
contracts/core/SiloGateway.sol
contracts/core/DebtManager.sol
```

From September 18th to September 21st the Cantina team conducted a review of Sturdyfi on commit hash e4b1e2cb. The team identified a total of **10** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 5
- Gas Optimizations: 0
- Informational: 4

3 Findings

3.1 Medium Risk

3.1.1 `DebtManager.manualAllocation` `vault.totalIdle` can be lower than `vault.minimum_total_idle()`

Severity: Medium Risk

Context: DebtManager.sol#L405

Description: In case the `vault.totalIdle()` is smaller the`vault.minimum_total_idle()` there should be a `continue`. The `totalIdle` can be lower than the minimum.

Recommendation:

```
  // deposit/increase not possible because minimum total idle reached
  if (position.debt > strategyData.current_debt &&
- vault.totalIdle() == vault.minimum_total_idle()) continue;
+ vault.totalIdle() <= vault.minimum_total_idle()) continue;
```

Sturdy: Addressed in commit 873c38a5.

Cantina: Fixed.

3.2 Low Risk

3.2.1 Permissionlessly realising strategy loss may be exploitable

Severity: Low Risk

Context: DebtManager.sol#L330-L332

Description: While `requestLiquidity()` is restricted to whitelisted gateways, the `SiloGateway`'s `borrowAsset()` is permissionless. Hence, `requestLiquidity()` is indirectly permissionless for JIT liquidity requests.

In the event that the selected strategy to pull liquidity from has unrealised losses, the loss will be realised immediately.

Thus, realising a vault's loss is permissionless and may possibly be exploited through the use of flash loans to create JIT liquidity requests. It could be the case that the strategy's loss is impermanent / temporary, and shouldn't be realised at that point in time (e.g. unrealised loss as a result of a temporal de-pegging of the asset).

Recommendation: Skip pulling liquidity from strategies with unrealised losses. It is safer for losses to be realised only with manual intervention. In this regard, also consider if realising losses through the zkVerifier should be allowed.

```
if (vault.assess_share_of_unrealised_losses(strategies[i], strategyDatas[i].current_debt - newDebt) != 0) {
-    vault.process_report(strategies[i]);
+    continue;
}
```

Sturdy: Addressed in commit b93936e4.

Cantina: Fixed.

3.2.2 The slippage parameter in requestLiquidity can result in blocking remaining requests

Severity: Low Risk

Context: DebtManager.sol#L352

Description: The requestLiquidity is used to request liquidity from other strategies in the vault.

In some edge cases (like losses) the vault.update_debt method used to change the liquidity doesn't update to the exact newDebt and it results in rounding issues. This means after the liquidity request steps requiredAmount wouldn't be equal to zero. Some DUST would remain in requiredAmount.

Therefore, the Sturdy team introduced a slippage parameter. Which is used to calculate a minAmount which is still tolerated.

```
uint256 minAmount = requiredAmount * _slippage / UTIL_PREC;
```

After, the liquidity request steps an amount smaller than the minAmount would still be tolerated.

```
require(requiredAmount < minAmount, Errors.AG_INSUFFICIENT_ASSETS);
```

In addition, an early stop condition has been added in case the requiredAmount gets smaller than the minAmount.

```
// early stop
if (requiredAmount < minAmount) break;
```

However, this early stop can also result in blocking required requests if the slippage is set too high.

Example: If we have two silos and one requesting silo together with a slippage of 1%:

- silo A → 1m liquidity available.
- silo B → 1m liquidity available.

The requestLiquidity function would revert if _amount > 1m && _amount < 1m + 10k, because the loop would stop after silo A in the early stop condition.

```
if (requiredAmount < minAmount) break;
```

The additional needed liquidity from silo B would not be requested.

Recommendation: Add an upper limit for the slippage parameter.

Sturdy: Addressed in commit 4a063dd5.

Cantina: Fixed. Max slippage settable is capped at 0.01%.

3.2.3 Sub-optimal strategy selection as oracle.getUtilizationInfo is based on global supply and borrows

Severity: Low Risk

Context: DebtManager.sol#L442

Description: The oracle.getUtilizationInfo function used in DebtManager._getAvailableAmountsAndDatas() returns the global borrow and supply. For example, the getUtilizationInfo of AaveV3AprOracle is defined as follows:

```
/**
 * @dev Get the current utilization info.
 * @param _strategy The strategy to get the utilization info of.
 * @return The current utilization info of the strategy.
 */
function getUtilizationInfo(
    address _strategy
) external view override returns (uint256, uint256) {
    address asset = IAaveV3Strategy(_strategy).asset();
    address aToken = IAaveV3Strategy(_strategy).aToken();
    DataTypesV3.ReserveData memory reserveData = LENDING_POOL.getReserveData(
        asset
    );

    uint256 availableLiquidity = IERC20(asset).balanceOf(aToken);
    uint256 totalDebt = IERC20Metadata(reserveData.stableDebtTokenAddress).totalSupply();
    totalDebt += IERC20Metadata(reserveData.variableDebtTokenAddress).totalSupply();

    return (totalDebt, totalDebt + availableLiquidity);
}
```

The `requestLiqiudity` function can only operate on supply deposited by the `vault` and can't access the global liquidity. This results in higher `availableAmounts` returned by `_getAvailableAmountsAndDatas`. In the `requestLiquidity` this would result in a higher `withdrawAmount`.

If `withdrawAmount >= current_debt => newDebt = 0` for the `vault.update_debt` function. This wouldn't result in a problem because only the available debt would be withdrawn. However, it may be a sub-optimal movement of funds, as the `vault` could have very high global liquidity, but very low liquidity deposited. The sorting would not result in the strategies with the highest available liquidity coming first.

Recommendation: Consider using the minimum of the strategy's `current_debt` and the calculated amount for the amounts returned by `_getAvailableAmountsAndDatas` function.

Sturdy: Addressed in commit dae43397.

Cantina: Fixed, as per the recommendation.

3.2.4 `DebtManager.setUtilizationTargetOfStrategy` doesn't check if `strategy` exists

Severity: Low Risk

Context: DebtManager.sol#L170

Description: The `setUtilizationTargetOfStrategy` doesn't check if the strategy exists before setting the `utilization` target.

Recommendation: Add a check to verify if the strategy exists.

Sturdy: Addressed in commit f8faabfb

Cantina: Fixed. The redundant `== true` equivalence check is removed in a subsequent commit when custom errors are used in lieu of require statements and revert strings.

3.2.5 `SiloGateway.setSlippage` can be set to more than 100%

Severity: Low Risk

Context: SiloGateway.sol#L37

Description: The `slippage` in the `SiloGateway` can be set to more than 100%.

Recommendation: Consider a reasonable upper limit to prevent an incorrect assignment.

Sturdy: Addressed in commit 64dce459.

Cantina: Fixed.

3.3 Informational

3.3.1 Improvements and cleanup recommendations

Severity: Informational

Context: DebtManager.sol

Description: This issue outlines a couple of small improvements that may be applied to the codebase.

Recommendation: Consider applying the recommendations listed below.

- **Redundant Import**

```
- import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

- **Redundant boolean comparison**

```
- require(strategyAvails[position.strategy] == true, Errors.AG_NOT_AVAILABLE_STRATEGY);
+ require(strategyAvails[position.strategy], Errors.AG_NOT_AVAILABLE_STRATEGY);
```

- **Spelling Error**

```
- SetZKVerfier
+ SetZKVerifier
```

- **Comment Improvement**

```
- // strategy -> bool, if the strategy is added ? true : 0
+ // strategy -> bool, if the strategy is added ? true : false
```

- **Rename** `minAmount` **to** `allowedSlippage` It's counter-intuitive of requiring the amount to be less than a minimum because of the variable name

```
require(requiredAmount < minAmount, Errors.AG_INSUFFICIENT_ASSETS);
```

```
- minAmount
+ allowedSlippage
```

- **Cache repeated calculation of** `requestingStrategyData.current_debt + _amount` The calculation is performed thrice in `requestLiquidity`. It can be saved in a variable `strategyNewDebt` instead (tested to not exceed stack size).

Sturdy: Addressed in commit 17c9d648.

Cantina: Fixed. All recommendations were adopted.

3.3.2 Cleanup `aprOracle` naming in `DebtManager`

Severity: Informational

Context: DebtManager.sol#L34

Description: `DebtManager` is still using the term `apr` in the `oracle` name.

Recommendation: Remove the term `apr` from the oracle, since the new approach is not based on `apr` anymore.

Sturdy: Addressed in commit 97367cd4.

Cantina: Fixed.

3.3.3 Usage of custom error types

Severity: Informational

Context: DebtManager.sol#L6

Description: Use custom error types instead of a library with string constants.

Recommendation: Instead of using the custom string library

```
library Errors {
    string internal constant AG_FEE_TOO_BIG = "1";
    //...
}

// usage
require(condition, Errors.AG_FEE_TOO_BIG);
```

Custom error types could be used as follows:

```
// define in contract
error AG_FEE_TOO_BIG();

// usage
if (!condition) revert AG_FEE_TOO_BIG();
```

Sturdy: Addressed in commit 03145881.

Cantina: Fixed.

3.3.4 `SiloManager.borrowAsset` allows to pass arbitrary `_collateralAsset` parameter

Severity: Informational

Context: SiloGateway.sol#L57

Description: The `borrowAsset` function doesn't verify if the passed `_collateralAsset` parameter is the actual collateral used by the silo.

Recommendation: Consider if the parameter can be derived from the silo. For instance, a FraxLend market has `collateralContract()`.

Sturdy: Addressed in commit a2c3b35c.

Cantina: Fixed. `_collateralAsset` is checked to be equivalent to the silo's collateral.