byterocket

Smart Contract Audit Report

Stroller Protocol

3rd of June 2022

Contents

Disclaimer

1. Preface

The developers of **Stroller Protocol** contracted byterocket to conduct a smart contract audit of their v0.1 smart contracts. Stroller Protocol "*allows users to earn yield on ERC20 tokens without the compulsion of having to wrap them to Super Tokens* [SuperFluid]. *This is enabled by creating Strollers which top-up the streams with just enough Super tokens to keep the streams running without any penalties*".

The team of byterocket reviewed and audited the above smart contracts in the course of this audit. We started on the 31st of May and finished on the 3rd of June 2022.

The audit included the following services:

- *Manual Multi-Pass Code Review*
- *Protocol/Logic Analysis*
- *Automated Code Review*
- *Formal Report*

byterocket gained access to the code via a public GitHub repository. We based the audit on the v0.1-prod branch's state on May 30th, 2022 (*commit hash 16e94b6d8a7d3f24b7dc6373f3c4ac7c2fb564b0*).

2. Manual Code Review

We conducted a manual multi-pass code review of the smart contracts mentioned in section (1). Three different people went through the smart contract independently and compared their results in multiple concluding discussions.

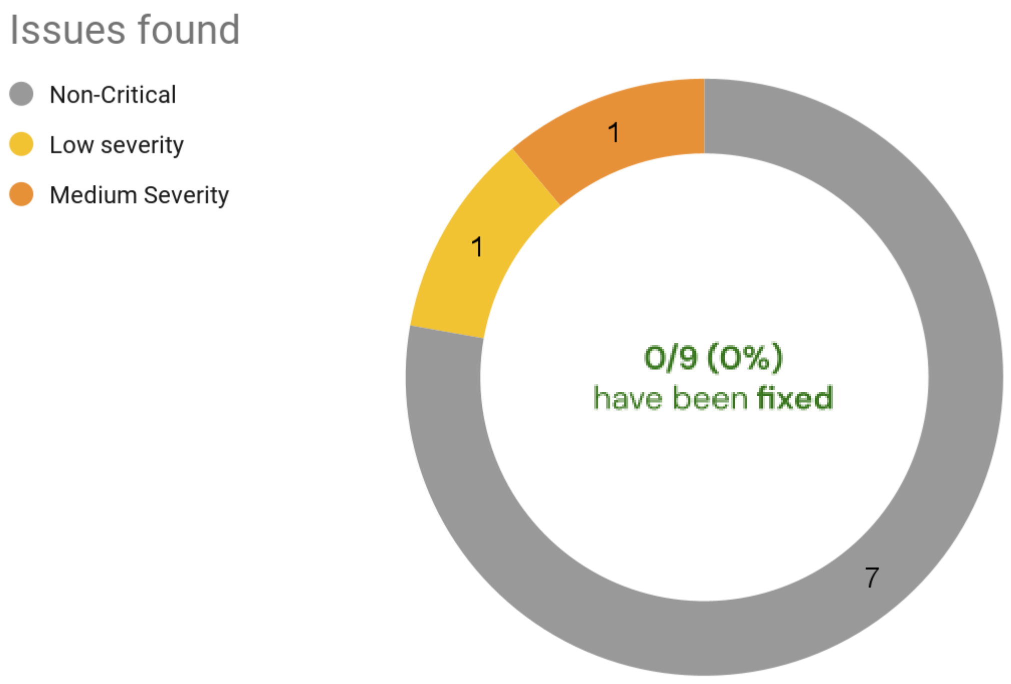
The manual review and analysis were additionally supported by multiple automated reviewing tools, like Slither, GasGauge, Manticore, and different fuzzing tools.

2.1 Severity Categories

We are categorizing our findings into four different levels of severity:

Non-Critical	Does not impose immediate risk but is relevant to security best practices. Includes issues with - Code style and clarity - Versioning - Off-chain monitoring
Low Severity	Imposes relatively small risks or could impose risks in the long-term but without assets being at risk in the current implementation. Includes issues with - State handling - Functions being incorrect as to specification - Faulty documentation or in-code comments
Medium Severity	Imposes risks on the function or availability of the protocol or imposes financial risk by leaking value from the protocol if external requirements are met.
High Severity	Imposes catastrophic risk for users and/or the protocol. Includes issues that could result in - Assets being stolen/lost/compromised - Contracts being rendered useless - Contracts being gained control of

2.2 Summary



On the code level, we **found 9 bugs or flaws, with** **1 of medium**, **1 of low severity**, **and 7 non-critical ones.** Additionally, we found **2 gas improvements**.

The contracts are written according to the latest standard used within the Ethereum community and the Solidity community's best practices. The naming of variables is very logical and understandable, which results in the contract being easy to understand. The code is sufficiently documented.

2.3 Findings

[MEDIUM SEVERITY] M.1 - Missing check on existence of index

Location: StrollManager.sol - Line 69 - 70

Description:
The index is the keccak256 hash of the users' address, the superToken address, and the liquidityToken address. Subsequently, the index is used to store the topUp of the user. In order not to overwrite anything, it is important to check whether there is already a topUp stored for the user, which is also indicated by the comment. However, this check is currently not implemented.

These are the affected lines of code:

```
// check if topUp already exists for given user and superToken
bytes32 index = getTopUpIndex(msg.sender, _superToken,
                              _liquidityToken);
```

Recommendation:
Consider adding a check to ensure that there is no topUp stored for this user already, e.g. via verifying like this

```
      if(topUp[index].user != address(0)) revert ...;
```

[LOW SEVERITY] L.1 - Incorrect error message for disabled strategy

Location: StrollManager.sol - Line 165 - 167

Description:
The required topUpAmount is determined by the checkTopUpByIndex() function, which returns zero in cases where e.g. the users balance or allowance doesn't suffice. Additionally, however, the case of a superToken being disabled for the corresponding strategy is handled in the same if-clause and returns the same zero output. Subsequently, it also triggers the TopUpNotRequired error, which in this case is not correct, as there might very well be a top-up required.

These are the affected lines of code:

```
uint256 topUpAmount = checkTopUpByIndex(_index);

if (topUpAmount == 0) revert TopUpNotRequired(_index);
```

Recommendation:
Consider handling the case of a `superToken` being disabled for the corresponding strategy differently, e.g. by also returning an error together with amount in the `checkTopUpByIndex()` function.

[NON CRITICAL] NC.1 - Add more checks to constructor

Location: StrollManager.sol - Line 27 - 35

Description:
There is no logic in the contract that actually enforces `minLower` to be less than `minUpper` and vice versa. As there is no way to change the values of `minLower` and `minUpper`, it should be ensured that they are correct. Additionally, if `minLower` is set to `0`, this would allow for unnecessary `0` TopUps by the Keepers to potentially happen.

These are the affected lines of code:

```
constructor(
  address _icfa,
  uint64 _minLower,
  uint64 _minUpper
) {
  CFA_V1 = IConstantFlowAgreementV1(_icfa);
  minLower = _minLower;
  minUpper = _minUpper;
}
```

Recommendation:
Consider adding the following checks
- `minLower < minUpper`
- `minLower > 0`

[NON CRITICAL] NC.2 - Do not emit events without state change

Location: StrollManager.sol - Line 122 - 131

Description:
Similar to the way the `removeApprovedStrategy()` function works, it is a common best practice to only emit events in cases where the state actually changed. This allows external tools to reconstruct the current state and all of the state changes without diving into the code or any specific transactions.

The same also applies to `strategies/StrategyBase.sol` in lines 15 - 25.

These are the affected lines of code:

```
function addApprovedStrategy(address _strategy) external override
        onlyOwner {
  if (_strategy == address(0)) revert InvalidStrategy(_strategy);

  approvedStrategies[_strategy] = true;
  emit AddedApprovedStrategy(_strategy);
}
```

Recommendation:
Consider changing the function to only emit the event if the strategy wasn't approved already, e.g. via something like this
`if (approvedStrategies[strategy]) return;`

[NON CRITICAL] NC.3 – Unnecessary cast to ISuperToken

Location: StrollManager.sol – Line 172

Description:
As `topUp.superToken` is already of type `ISuperToken` and not an address, the cast is unnecessary.

These are the affected lines of code:

```
ISuperToken(topUp.superToken)
```

Recommendation:
Consider not unnecessarily casting `topUp.superToken` to `ISuperToken`.

[NON CRITICAL] NC.4 – Missing a helpful zero check

Location: strategies/AaveV2StrollOut.sol – Line 43

Description:
Generally it is advisable to properly verify return values early, especially if they are used for subsequent calls without a potential error quickly appearing. In this case, an `underlyingToken` with the zero address in line 43 would lead to a lot of unnecessary calls and computations that could have been prevented.

The same also applies to `strategies/ERC20StrollOut.sol` in line 29.

These are the affected lines of code:

```
IERC20Mod underlyingToken = IERC20Mod(_superToken.getUnderlyingToken());
```

Recommendation:
Consider ensuring that `underlyingToken != address(0)`.

[NON CRITICAL] NC.5 - Not using SafeERC20 transfers

Location: strategies/AaveV2StrollOut.sol - Line 86 - 87

Description:
In the `topUp()` function, `superTokens` are transferred to the user without making use of the `SafeERC20` variant even though it's imported and used for other transfers. This might be the case as `superTokens` can be trusted with their implementation being very familiar with the developers, but it would still be consistent to use `SafeERC20` for any transfers.

The same also applies to `strategies/ERC20StrollOut.sol` in lines 62 - 63.

These are the affected lines of code:

```
if (!_superToken.transfer(_user, adjustedAmount))
  revert TransferFailed(_user, address(_superToken), adjustedAmount);
```

Recommendation:
Consider also using `safeTransfer` for superTokens as well.

[NON CRITICAL] NC.6 - Don't use change-function for initialization

Location: strategies/StrategyBase.sol - General

Description:
In the `StrategyBase` contract, there is a `changeStrollManager()` function, which is seemingly also used to initialize the value of the `strollManager` variable. According to the best practice, this should be done via the `constructor`.

Recommendation:
Consider adding the `constructor` method to set the initial value of the `strollManager` variable.

[NON CRITICAL] NC.7 - Use try-catch to control error message

Location: strategies/StrategyBase.sol - Line 31 - 32

Description:
In the `emergencyWithdraw()` function, arbitrary tokens can be transferred out of the contract in case they got stuck, etc. As errors can also occur in the ERC20 contract itself

during the `transfer()` call, the custom `TransferFailed` error may not be emitted accordingly.

As an additional remark, we want to point out that using `safeTransfer()` instead of `transfer()` would make a lot of sense here, as downstream contracts use it anyways and subsequently not using it would not save any gas cost otherwise.

These are the affected lines of code:

```
if (!IERC20Mod(_token).transfer(msg.sender, tokenBalance))
  revert TransferFailed(msg.sender, _token, tokenBalance);
```

Recommendation:
Consider making use of a `try-catch`-block to ensure that the custom `TransferFailed` error is thrown each time a transfer failed due to any reason.

Also consider using `safeTransfer()` instead of a regular `transfer()`.

2.4 Gas Optimizations

[Gas Optimization] GO.1 - Optimize loops to be more efficient

Description:
All of the for-loops in the project are using the standard way of declaring them, which is slightly inefficient gas-wise.

The following loops are not optimized:
- StrollManager.sol - Line 116

This is an example of an unoptimized for-loop:

```
for (uint256 i = 0; i < array.length; i++) {
  [...]
}
```

Recommendation:
Consider optimizing the for-loops in each contract to be more efficient in terms of its gas usage, including:
- Not initializing variables with their default value
- Using the more efficient way to increment a variable

```
for (uint256 i; i < array.length; ++i) {
  [...]
}
```

[Gas Optimization] GO.2 - Cache certain variables to save gas

Description:
In certain circumstances, it saves gas to cache variables that are read often, especially if they are stored in storage and not in memory.

The affected variables are:
- StrollManager.sol - Line 116 (`_indices.length`)

Recommendation:
Consider caching the affected variables and reading from the cached version instead of doing costly read actions.

3. Protocol/Logic Review

Part of our audits are also analyses of the protocol and its logic. The byterocket team went through the implementation and documentation of the implemented protocol.

The repository itself contained tests and documentation. We found the provided unit tests that are coming with the repository execute without any issues and cover the most important parts of the protocol.

According to our analysis, the protocol and logic are working as intended, given that any findings with a severity level are fixed.

We were **not able to discover any additional problems** in the protocol implemented in the smart contract.

4. Summary

During our code review (*which was done manually and automated*), we **found 9 bugs or flaws, with 1 of medium, 1 of low severity, and 7 non-critical ones.** Our automated systems and review tools did **not find any additional ones**. Besides any bugs or flaws, we found **2 gas improvements**.

The protocol review and analysis did neither uncover any game-theoretical nature problems nor any other functions prone to abuse besides the ones that have been uncovered in our findings.

In general, there are some improvements that can be made, but we are **happy** with the overall quality of the code and its documentation. The developers have been very responsive and were able to answer any questions that we had.