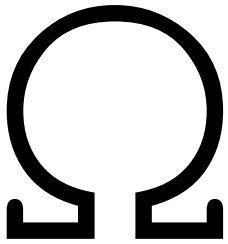
Backed Rebasing Token Solidity Contracts

Final Audit Report

July 2, 2024



Team Omega

Teamomega.eth.limo

Summary 2
Scope of the Audit 3
Methods Used 3
Resolution 4
Disclaimer 4
Severity definitions 4
Findings 6
General 6
Centralization risks 6
BackedAutoFeeTokenFactory.sol 7
BF1. proxyAdmin can be marked immutable [info] [resolved] 7
BackedAutoFeeTokenImplementation.sol 8
BI1. delegatedTransferShares may transfer wrong amount [high] [resolved] 8
BI2. setPeriodLength and setLastTimeFeeApplied should update the multiplier before updating the period length [low] [resolved] 8
BI3. Contract initialize function might be re-called by anyone [low] [resolved] 8
BI4. Remove unused import [info] [resolved] 9
BI5. Declare functions as external instead of public when possible [info] [resolved] 9
BI6. Wrong visibility declaration for _initialize_auto_fee [info] [resolved] 9
BI7. Declare functions as pure when possible [info] [resolved] 9
BI8. Use helper functions instead of duplicating calculation code [info] [resolved] 10
BI9. __gap variable can be removed [info] [resolved] 10
BI10. Disable initializers on the implementation contract [info] [resolved] 10

Summary

Backed Finance has asked Team Omega to audit the contracts that define the behavior of a rebasing token and factory.

We found **1 high severity issue** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **no** issues as “medium” - these are issues we believe you should definitely address. We did classify **2** issues as “low”, and an additional **8** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

All these issues were subsequently resolved.

Severity	**Number of issues**	**Number of resolved issues**
High	1	1
Medium	0	0
Low	2	2
Info	8	8

Scope of the Audit

The audit concerns files and changes developed in the following pull request under the contracts folder

- `https://github.com/backed-fi/backed-token-contract/pull/32`

The audit report was based on the following commit:

```
655fdd415f53956ae28e67608a1b8582a0a90351
```

Which was an update of `25333f30d55d88ce397166e120961591984148ec` and adds a number of new files:

```
BackedAutoFeeTokenFactory.sol
BackedAutoFeeTokenImplementation.sol
BackedFactory.sol
```

The `WrappedBackedToken.sol`file is not in scope of this report.

Methods Used

The contracts were compiled, deployed, and tested in a test environment.

Code Review

We manually inspected the source code to identify potential security flaws.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues were found, and we have included them below in the appropriate parts of the report.

Resolution

The issues were subsequently addressed in commit:

```
b317bdf1ff9f9db0dfef0407c99bf6f7b1acf12f
```

We checked the fixes and marked the resolution of the issues below. All issues were resolved, except two issues that were categorized as info, and do not represent a vulnerability.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

Centralization risks

The Backed token contracts are centrally managed contracts, and define a number of roles that control parameters that can make the contract unusable, and can arbitrarily inflate or deflate the token balances.

This centralization is by design, and by itself does not constitute an unacceptable security risk. But these roles must be carefully managed.

It is important, for both users of the contracts as well as the Backed team, to be aware of what can go wrong if an admin account is compromised, or if an admin makes mistakes. The following table lists the various admin roles, what they control, and the kind of risks that follow from these controls.

role	controls	risks
proxyAdminOwner	Can upgrade the contract	The owner of the proxyAdmin can change the behavior and all balances of the contract. It can brick the contract.
tokenOwner	updateFeePerPeriod setLastTimeFeeApplied setPeriodLength sactionsList delegationList Role assignments of minter, burner, pauser and multiplierUpdater	Can set the fees to any value (also to 100%, zero-ing out the balances). Can brick the contract by setting the fee > 1e18. If lastTimeFeeApplied is far in the past, the contract may become unusable (almost all actions will revert because of out of gas errors). If the variable is set to a date in the future, most actions will revert. If this value is set to a very low value, there is a risk that actions run out of gas and the contract becomes unusable The account can circumvent the sanction list If these roles are set to compromised accounts, damage can be done
minter	Can mint new tokens to any account	Can mint new tokens to any account
burner	Can burn its own tokens, and the tokens held by the token contract	No specific risks
pauser	Can pause the contracts	Economic damage if the token becomes temporarily untradeable
multiplierUpdater	Can set the multiplier to any value	The multiplierUpdater can arbitrarily inflate or deflate the (total) token balance and balance of all accounts. If the multiplier is set to a very large value, the contract becomes unusable

We recommend that the Backed team take the usual precautions, where compatible with their requirements: assign these roles to TimeLock contracts that are controlled by a multisig.

As for all rebasing tokens where the “rebase factor” is not necessarily continuous in time, special attention must be taken when taking actions that make (large) sudden changes in the balances of the users - in this case, abrupt balance changes can occur when changing the multiplier, periodLength, lastTimeFeeapplied and feePeriod. (For example, a user may place a trade for prices from just before a scheduled update of the multiplier (say to reflect a “stock split”), that gets executed after the multiplier is updated (and so the price has significantly changed). In such cases, it can make sense to pause trading for a limited period in time.

BackedAutoFeeTokenFactory.sol

BF1. proxyAdmin can be marked immutable [info] [resolved]

The `proxyAdmin` variable is set in the constructor and never changes, which means it can be marked as immutable.
Recommendation: Mark the `proxyAdmin` variable as immutable.
Severity: Info
Resolution: The issue was resolved as recommended.

BackedAutoFeeTokenImplementation.sol

BI1. delegatedTransferShares may transfer wrong amount [high] [resolved]

The `delegatedTransferShares` function doesn't call `updateMultiplier` before calculating the amount of tokens to transfer, which means it uses the stored value of the multiplier to calculate the amount. However, the function calls `updateMultiplier` as part of its internal logic, when it calls `_beforeTokenTransfer`, which means that, if the multiplier has changed, when calculating back the amount of shares to transfer the result will be different than the original amount of shares to send.
Recommendation: Add the `updateMultiplier` modifier to the `delegatedTransferShares` function.
Severity: High
Resolution: The issue was resolved as recommended.

BI2. setPeriodLength and setLastTimeFeeApplied should update the multiplier before updating the period length [low] [resolved]

The `setPeriodLength` and `setLastTimeFeeApplied` functions update the period length for updating the multiplier, but to avoid affecting past periods, they should first call `updateMultiplier` to update it

for the periods that have already past, otherwise a situation of race condition may arise between the call to these functions and any call that triggers the `updateMultiplier` modifier.
Recommendation: Add the `updateMultiplier` modifier to the `setPeriodLength` and `setLastTimeFeeApplied` functions.
Severity: Low
Resolution: The issue was resolved as recommended.

BI3. Contract initialize function might be re-called by anyone [low] [resolved]

The `initialize` and `initialize_v2` functions in the contract are meant to be called only once on deployment/ upgrade, and are protected by a check that the `lastTimeFeeApplied` is set to 0. Yet there is no check in the `initialize` or `setLastTimeFeeApplied` functions that the value given to the `lastTimeFeeApplied` isn't 0, in which case re-calling the initialize function will be possible by anyone.
Recommendation: Add a requirement in the `_initialize_auto_fee` and `setLastTimeFeeApplied` functions that the new value for `lastTimeFeeApplied` is not 0.
Severity: Low
Resolution: The issue was resolved as recommended.

BI4. Remove unused import [info] [resolved]

The contract imports the `SanctionsList.sol`, but does not use it and it could be removed.
Recommendation: Remove the unused import.
Severity: Info
Resolution: The issue was resolved as recommended.

BI5. Declare functions as external instead of public when possible [info] [resolved]

The contract contains various `public` functions which are meant to be accessed only externally, and can therefore be marked as `external` instead.
Recommendation: Mark functions as `external` when possible.
Severity: Info
Resolution: The issue was resolved as recommended.

BI6. Wrong visibility declaration for _initialize_auto_fee [info] [resolved]

The `_initialize_auto_fee` function is meant to be an `internal` function, but is declared as `public`.
Recommendation: Mark the `_initialize_auto_fee` function as `internal`.
Severity: Info
Resolution: The issue was resolved as recommended.

BI7. Declare functions as pure when possible [info] [resolved]

The `_getSharesByUnderlyingAmount` and `_getUnderlyingAmountByShares` functions are declared as `view`, but since they don't read from the contract storage they can more appropriately be declared as `pure`.
Recommendation: Mark the `_getSharesByUnderlyingAmount` and `_getUnderlyingAmountByShares` functions as `pure`.
Severity: Info
Resolution: The issue was resolved as recommended.

BI8. Use helper functions instead of duplicating calculation code [info] [resolved]

The `totalSupply` and `balanceOf` functions calculate token amounts based on the shares and multiplier, but the calculation they perform is duplicated from the `_getUnderlyingAmountByShares` helper function. It is recommended to avoid such duplication and instead use the helper function to perform the calculation.
Recommendation: Use `_getUnderlyingAmountByShares` in `totalSupply` and `balanceOf`.
Severity: Info
Resolution: The issue was resolved as recommended.

BI9. __gap variable can be removed [info] [resolved]

Since the contract is not being inherited by any other contract, there is no need currently for it to have a `__gap` variable to reserve storage slots.
Recommendation: Remove the `__gap` variable from the contract.
Severity: Info
Resolution: The issue was resolved as recommended.

BI10. Disable initializers on the implementation contract [info] [resolved]

For consistency, you could disable the `initialize_v2` function on the implementation contract by setting the value of `lastTimeFeeApplied` in the constructor.
Recommendation: Set the `lastTimeFeeApplied` value in the constructor to be non-zero.
Severity: Info
Resolution: The issue was resolved as recommended.