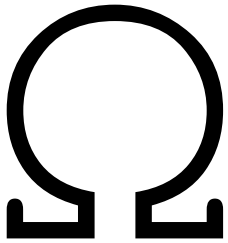
Wrapped Backed Solidity Contracts

Audit Report - October 15, 2023
Updated on January 30, 2024



Team Omega

Teamomega.eth.limo

Summary

Backed Finance has asked Team Omega to audit the contracts that define the behavior of a TokenWrapper with whitelist functionality.

Omega has audited the Backed Finance ERC20 Token in May 2022. The present report covers changes that were made after that date.

We found **no high severity issues** - 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 **1** issue as “low”, and an additional **12** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Backed Finance subsequently implemented fixes for a large number of the issues. We reviewed those fixes, and included the resolution in the report.

Severity	**Number of issues**	**Number of resolved issues**
High	0	0
Medium	0	0
Low	1	0
Info	15	12

Scope and resolution of the Audit

The audit concerns files developed in the following repository under the contracts folder:

- `https://github.com/backed-fi/wrapped-tokens/`

We audited commit `5ffbd7e7b694f0ecc9d90a938ad3011834dbd130` in October 2023, and noted the issues listed below. Backed addressed most of the issues. We checked the changes and updated the report. This resulted in commit `50bcea3c408ab2536ac3041430b7c44ec52282fc`.

In January 2024 an integration with KeyRing was added to the code. We reviewed these additions, and noted some small issues, which were all addressed in the following commit

`0d92facd936163d2982a8a06a2f11c3966f31255`

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.

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

G1. Insufficient test coverage [info] [resolved]

The test coverage is only covering 84% of lines, and less than 50% of branches. Also it includes the mock file which should be ignored in the configurations.
Recommendation: It is recommended to reach a 100% test coverage. Also, you should configure the coverage script to ignore the mock folder.
Resolution: The issue was resolved as recommended.

G2. Pin versions of solidity dependencies [info] [resolved]

In `package.json`, a possible range of versions of the OpenZeppelin dependencies is specified rather than a single one:

```
"@openzeppelin/contracts": "^4.9.3",
"@openzeppelin/contracts-upgradeable": "^4.9.3",
```

This can lead to unexpected problems when a new version of OpenZeppelin is released and other developers (or the continuous integration process, or yourself at a later date) will recompile the contracts with this new version (for example to verify the compiled code on etherscan)
Recommendation: Specify fixed versions of smart contract dependencies in `package.json`, instead of ranges, so that the solidity code can be verified and there is no ambiguity about the actual code you are to deploy or already have deployed.
Severity: Info
Resolution: The issue was resolved as recommended.

G3. Use revert with custom errors instead of require [info] [resolved]

To revert, the contracts currently use `require` statements with reason strings. It would save gas to use the newer syntax of revert with custom messages.
Recommendation: Replace the use of `require` statements with `revert` with an Error object.
Severity: Info
Resolution: The issue was resolved as recommended.

G4. Declare functions as external instead of public when possible [info] [resolved]

All `public` functions that are declared in `WhitelistController`, `WhitelistControllerAggregator`, and `WrappedBackedToken` are not used within the contracts and can be declared `external`.
Recommendation: Mark all `public` functions of the mentioned files as `external`.
Severity: Info
Resolution: The issue was resolved as recommended.

G5. Remove unused imports [info] [resolved]

All contracts audited contain unused imports that should be removed.

In `WhitelistController.sol` the following import statements can be removed:

```
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import
"@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20WrapperUpgradeable.sol
";
import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

In `WhitelistControllerAggregator.sol` these imports are unused:

```
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import
"@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20WrapperUpgradeable.sol
";
import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

In `WrappedBackedToken.sol` these imports are unused:

```
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "./WhitelistControllerAggregator.sol";
```

In `WrappedBackedTokenFactory.sol` these imports are unused:

```
import "@openzeppelin/contracts/governance/TimelockController.sol";
```

Recommendation: Remove all unused imports.
Severity: Info
Resolution: The issue was resolved as recommended.

G6. Call _disableInitializers in the constructor of proxy contracts [info] [resolved]

The WrappedBackedToken calls `_disableInitializers` in its constructor, while the `WhitelistController` and `WhitelistControllerAggregator` do not. For consistency and best practices, it's recommended to use it in those contracts as well.
Recommendation: Add a constructor which calls `_disableInitializers` in the `WhitelistController` and `WhitelistControllerAggregator` contracts.
Severity: Info
Resolution: The issue was resolved as recommended.

G7. Add a license file and claim copyright [info] [resolved]

The solidity files are marked with a license identifier:

```
// SPDX-License-Identifier: MIT
```

However, the repository does not contain a copy of the MIT license.
The MIT license works by claiming the copyright of the code yourself, and then granting usage to third parties to use your software.
Recommendation: Add the MIT license text from https://opensource.org/license/mit/ to the repository and edit it with your data.
Severity: Info
Resolution: The issue was resolved as recommended.

WhitelistController.sol

WC1. Variable whitelist missing explicit visibility declaration [info] [partially resolved]

The `whitelist` variable is missing an explicit visibility declaration. Also, it has a getter called `isWhitelisted`, which could be obsoleted by making `whitelist` a public variable (you could rename it to `isWhitelisted` so the getter name remains the same).
Recommendation: Mark the `whitelist` variable as public and remove its getter.
Severity: Info
Resolution: The issue was partially resolved. The `whitelist` variable is now explicitly declared as public, but still has a getter in the contract which means it will now have 2 getters. It would be better to remove the getter or mark the variable as internal to avoid the duplication.

WC2. Add the option update whitelisting of multiple addresses in a single call [info] [resolved]

Currently, the only way to update the whitelist status of multiple addresses is to make multiple calls to the contract, yet it could save gas to allow passing multiple addresses which will be updated together in a single call.
Recommendation: Add the option to pass update the whitelist status of multiple addresses at once.
Severity: Info
Resolution: The issue was resolved as recommended.

WrappedBackedToken.sol

WBT1. Variable whitelistController should be WhitelistControllerAggregator [low] [not resolved]

The contract defines a `whitelistController` variable of type `WhitelistController` as its whitelisting manager. However, the intended use here is to use an instance of `WhitelistControllerAggregator` as their whitelisting manager. This is also how the factory deploys new wrapped token contracts.
This type definition will work with the current interface, because `WhitelistController` and `WhitelistControllerAggregator` happen to both implement the `isWhitelisted` function with the same parameters. But this may change in future versions.
Recommendation: Change the `whitelistController` to be a `WhitelistControllerAggregator`.
Severity: Low
Resolution: The issue was not resolved.

WBT2. Unnecessary inheritance of Initializable [info] [resolved]

The contract inherits from `Initializable`, but also from other contracts which inherit `Initializable`, so `Initializable` does not need to be listed explicitly.
Recommendation: Remove the explicit inheritance of `Initializable`.
Severity: Info
Resolution: The issue was resolved as recommended.

WBT3. Tokens of de-whitelisted addresses are stuck in the wrapper contract [info] [not resolved]

If an address that has a token balance is removed from the whitelist, there will be no way of moving the tokens out of that user's balance, and the tokens will be stuck in the contract (unless the address is whitelisted again).
Recommendation: Depending on what the requirements are, this may either be desirable, or not at all. Consider creating a method with which a designated account (for example, the owner of the contract) can withdraw tokens from non-whitelisted addresses. Or, again if compatible with your requirements, you could consider applying the whitelist only to the destination addresses (i.e. any address can send tokens, but only whitelisted addresses can receive tokens).
Severity: Info

Resolution: The issue was not resolved.

WrappedBackedTokenFactory.sol

WBTF1. WrappedBackedTokenProxy constructor does not need to be payable [info] [resolved]

The WrappedBackedTokenProxy has a `payable` constructor, but does not need to accept ETH and so the `payable` can be removed.
Recommendation: Remove the payable from the `WrappedBackedTokenProxy` constructor.
Severity: Info
Resolution: The issue was resolved as recommended.

WBTF2. proxyAdmin can be marked immutable [info] [resolved]

The `proxyAdmin` variable can be marked `immutable`, or a setter could be added if you'd like to keep its value changeable.
Recommendation: Either mark the `proxyAdmin` as `immutable` or add a setter to allow the owner to set its value.
Severity: Info
Resolution: The issue was resolved as recommended, `proxyAdmin` is now marked as `immutable`.

WBTF3. Syntax to get function selector could be simplified [info] [resolved]

The function selector for the initialize functions of contracts is retrieved using the following syntax: `WhitelistControllerAggregator(address(0)).initialize.selector`, yet the `(address(0))` is unnecessary and can be removed.
Recommendation: Remove the `(address(0))` when getting the function selectors.
Severity: Info
Resolution: The issue was resolved as recommended.

WBTF4. Cannot deploy wrapper for the same token with different token owners [info] [not resolved]

The `deployWrappedToken` uses only the underlying token address for the salt that is used to generate the contract address, this means it will not be possible to deploy multiple wrappers, for example with different sets of whitelisted token owners, for the same token. Depending on the use case, this option may be desirable.
Recommendation: If this is intended the code can be left unchanged, however, if deploying multiple wrappers for the same token is needed in certain cases, then the salt generation should be modified.
Severity: Info
Resolution: The issue was not resolved.