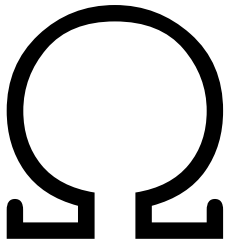
Backed Token Bridge

Final Audit Report

First report: September 20, 2024
Final Report: October 14, 2024



Team Omega

Teamomega.eth.limo

Summary 2
Scope of the Audit 3
Resolution 3
Methods Used 3
Disclaimer 4
Severity definitions 4
Findings 5
General 5
G1. Copyright should be claimed by Backed [low] [resolved] 5
G2. Pin solidity dependencies in package.json [low] [resolved] 5
G3. Use a more recent version of solidity [info] [resolved] 6
G4. Remove unused dependencies in package.json [info] [resolved] 6
BackedCCIPReceiver.sol 6
B1. Store allowed sender addresses together with their source chain [low] [resolved] 6
B2. Do not revert when trying to send invalid messages [low] [resolved] 7
B3. Store allowed sender addresses as bytes instead of address [info] [resolved] 7
B4. Mark functions that are not used internally as external [info] [resolved] 8
B5. Dependency on Initializable can be removed [info] [resolved] 8
B6. add tokenReceiver to the MessageSent event [info] [resolved] 8
B7. Call setters in initialize [info] [resolved] 8
CCIPReceiverUpgradeable.sol 9
C1. Add a __gap variable [low] [resolved] 9
C2. i_ccipRouter should be private [info] [resolved] 9

Summary

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

We found **no 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 **5** 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.

Severity	Number of issues	Number of resolved issues
High	0	0
Medium	0	0

Low	5	5
Info	8	8

Scope of the Audit

Scope of the Audit
The audit concerns an upgrade of the Backed token contracts with rebasing functionality.
As of writing, the code is still under development in the following repository and branch:

```
https://github.com/backed-fi/backed-ccip-contract
```

The code to be audited are two files that are adapted from the ChainLink's example code, and concerns two files:

```
contracts/ccip-upgradeable/CCIPReceiverUpgradeable.sol
contracts/BackedCCIPReceiver.sol
```

The audit report was based on the following commit:

```
dfd64ae580eedcaead0789b696e947556f0cbd2c
```

Resolution

The issues we found were addressed in following commit, which also adds some new features:

```
05e0ea7d87d914f02cca9acf6bf64056eddf8299
```

We reviewed all the changes.

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. We also inspected the context in which the contracts were developed, namely the Chainlink Bridge architecture.

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. Copyright should be claimed by Backed [low] [resolved]

The software includes a copy of the MIT license, but the copyright is not claimed by the authors, nor does it have the correct year:

```
Copyright (c) 2023 SmartContract
```

Recommendation: Backed needs to claim the copyright to release the software under the MIT license to activate the license.
Severity: Low
Resolution: The issue was resolved as recommended

G2. Pin solidity dependencies in package.json [low] [resolved]

It is desirable to have a deterministic outcome when compiling smart contracts. Currently, `package.json` specifies a range for the chainlink dependency:

```
"@chainlink/contracts-ccip": "^1.4.0",
```

This will make it so that when a newer version becomes available, a fresh build of their repository, they will use this newer version, resulting in different bytecode for the compiled contracts, which makes it hard to verify the deployment of the contracts, and may possibly create issues with backward compatibility.
Recommendation: Pin a specific version of the dependency

```
"@chainlink/contracts-ccip": "1.4.0",
```

Severity: Low
Resolution: The issue was resolved as recommended.

G3. Use a more recent version of solidity [info] [resolved]

`hardhat.config.js` defines the solidity version with which the contracts will be compiled and deployed:

```
solidity: "0.8.19",
```

This is not the latest version.
Recommendation: Consider using a more recent version (as of writing, the current release is 0.8.23 - https://github.com/ethereum/solidity/releases).
Severity: Low
Resolution: The issue was resolved as recommended.

G4. Remove unused dependencies in package.json [info] [resolved]

The following dependency in package.json is not used and can be removed:

```
"@chainlink/contracts"
```

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

BackedCCIPReceiver.sol

B1. Store allowed sender addresses together with their source chain [low] [resolved]

The `allowlistedSenders` mapping stores addresses that are allowed to send messages from any of the chains listed in `allowlistedSourceChains`.

Generally speaking, it is possible that addresses on different chains are controlled by different EOA accounts (e.g. a multisig may have different owners on different chains), and in certain (rare) cases an address that is controlled by a trusted party may be taken by an attacker on another chain (this happened: https://rekt.news/wintermute-rekt/).

For the intended use of of the whitelist in the current application, where the allowed senders will be instances of `BackedCCIPReceiver`, it will probably not be the case that the deployment addresses

of these contracts will be the same on different chains - so there is no reason to keep a chain-agnostic whitelist.

Recommendation: Store the allowed sender address on a per-chain basis, i.e. define it as a mapping from chainIds to allowed senders.

```
      // mapping from source chains to allowed senders on that chain
       mapping(uint64 => (address => bool)) public
allowlistedSenders;
```

Severity: Low. The probability of abuse is very small, as the whitelist will (probably) contain just a single `BackedCCIPReceiver` for each supported chain as an allowed sender address.
Resolution: The issue was resolved as recommended - the contract now uses:

```
      mapping(uint64 => address) public allowlistedSourceChains;
```

B2. Do not revert when trying to send invalid messages [low] [resolved]

The `send` function reverts if an address is not in the allowlist. The effect will be that the message in the `EVM2EVMOffRamp` contract will be set to `FAILURE`. This means that the message remains in the queue, and users can try to execute it again and again. (It will, of course, continue to fail).
It would be more elegant to only set messages in `FAILURE` execution state for which you want execution to be retried. (For example, messages that fail because the custody wallet does not hold enough funds). Payments that are not executed because messages are incorrect (for example, if they originate from a non-whitelisted address) should simply return - in that way the execution state message will be marked as `SUCCESS` and will be removed from the queue.
Recommendation: The `send` function should only revert in cases where you want the message to be retried.
Severity: Low
Resolution: The issue was resolved as recommended.

B3. Store allowed sender addresses as bytes instead of address [info] [resolved]

The mapping of `allowListedSenders` stores addresses of senders that are allowed to initiate messages. The value that is being checked is `any2EvmMessage.sender`, which is of type `bytes`. The reason for that is to allow for addresses from a non-EVM chain. You could change to store bytes instead of addresses to avoid the need for conversion and save gas.
Recommendation: Use the `bytes` type instead of `address` to store addresses in `allowListedSenders`. This saves gas as conversions between bytes and addresses are not necessary, and keeps the implementation generic.

Severity: Low
Resolution: Our suggestion was not taken.

B4. Mark functions that are not used internally as external [info] [resolved]

The functions `withdraw`, `withdrawTokens` and `getDeliveryFeeCost` are currently marked as `public`, but are unused in the contract and so can be marked `external`.
Recommendation: The functions can be marked as `external` to save some gas costs.
Severity: Low
Resolution: The issue was resolved as recommended.

B5. Dependency on Initializable can be removed [info] [resolved]

There is no need to inherit from `Initializable`, as `CCIPReceiverUpgradeable` already does.
Recommendation: Remove superfluous dependencies.
Severity: Info
Resolution: The issue was resolved as recommended.

B6. add tokenReceiver to the MessageSent event [info] [resolved]

You may find it useful to add the `tokenReceiver` (i.e. the final destination of the tokens) to the `MessageSent` event.
Severity: Info
Resolution: The issue was resolved as recommended.

B7. Call setters in initialize [info] [resolved]

In the `initialize` function, values for the custody wallet and the gas costs are set directly:

```
_custodyWallet = _custody;
_defaultGasLimitOnDestinationChain = _gasLimit;
```

Which means the setter events won't be emitted for their initial value.
Recommendation: Use the corresponding setters, so events are emitted.
Severity: Info
Resolution: The issue was resolved as recommended.

CCIPReceiverUpgradeable.sol

C1. Add a __gap variable [low] [resolved]

The `CCIPReceiverUpgradeable` contract is part of an inheritance chain of upgradeable contracts, and it is good practice to add a `__gap` variable to make it easier to add state variables to the contract in any future upgrades.

Recommendation: Add a `__gap` variable to reserve up to 50 storage slots for this contract:

```
uint256[49] private __gap;
```

Severity: Low
Resolution: The issue was resolved as recommended.

C2. i_ccipRouter should be private [info] [resolved]

In the original implementation, `i_ccipRouter` is marked `immutable`, but since now the contract is upgradeable, the `immutable` keyword was removed. However, this means that contracts which inherit the `CCIPReceiverUpgradeable` may now modify the value of `i_ccipRouter`, which does not seem intended.
Recommendation: Make the `i_ccipRouter` variable as `private` to reflect correctly its intended visibility.
Severity: Info
Resolution: The issue was resolved as recommended.