PUBLIC

Code Assessment

of the USDT0

Smart Contracts

14 January, 2024

Produced for

USD₮0

by

CHAINSECURITY

Contents

1 Executive Summary 3

2 Assessment Overview 5

3 Limitations and use of report 7

4 Terminology 8

5 Findings 9

6 Resolved Findings 10

7 Notes 12

1 Executive Summary

Dear USDT0 Team,

Thank you for trusting us to help with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of USDT0 according to Scope to support you in forming an opinion on their security risks.

USDT0 offers a set of smart contracts that implement the Omnichain extension for USDT. These smart contracts rely on LayerZero's provided Omnichain Fungible Token (OFT) infrastructure to facilitate bridging functionalities to other chains.

The most critical subjects covered in our audit are asset solvency, cross-contract interaction and access control. Security regarding all aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security. Discovered low-severity issues do not pose an immediate treat and can only be triggered by human error. They were addressed in Version 2 of the code; however redeployment was considered unwarranted. If bugs ever trigger, they can be mitigated by upgrading the system.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2
• Code Corrected	1
• Risk Accepted	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the following source code files inside the USDT0 repository:

OFT contracts

- contracts/OAdapterUpgradeable.sol
- contracts/OUpgradeable.sol

Tether OFT Extension

- contracts/Wrappers/OFTExtension.sol

The table below indicates the code versions relevant to this report and when they were received.

OFT contracts

V	Date	Commit Hash	Note
1	05 January 2025	a53ae5822b71a091ab07decccbf4f3801965d31b	Initial Version

Tether OFT extension

V	Date	Commit Hash	Note
1	05 January 2025	0c376cf55da2acf0ad748d6b0bf4933a92888c07	Initial Version
2	14 January 2025	8b5542392f5b40873e2779c6f6b1caf3b829fcf3	Version with fixes

For the Solidity smart contracts, the compiler version `0.8.4` was chosen for Tether OFT Extension and the floating version `^0.8.22` was chosen for OFT contracts.

2.1.1 Excluded from scope

All smart contracts not explicitly mentioned in the scope section are excluded from the assessment. This includes all test code, deployment scripts, mocks and libraries. LayerZero contracts are also excluded from the scope of the audit and assumed to function according to their specifications. Potential upgrades to contracts that support upgradability are also excluded from the scope of the audit.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

USDT0 is Tether Token USD, that can be bridged to other blockchains using LayerZero Omnichain infrastructure.

To achieve this, on Ethereum mainnet, `OAdapterUpgradeable` contract will be deployed. This contract extends functionality of LayerZero's OFT Adapter contract. When tokens are bridged to other chains, `USDT` tokens will be transferred to this contract and locked. When bridged back from other chains, tokens will be unlocked and transferred back to specified receiver.

On Ink mainnet, a pair of contracts will be deployed: `TetherTokenOFTExtension` and `OUpgradeable`. `TetherTokenOFTExtension` is an ERC20 token, which extends the functionality of Tether Token V2. This extension allows only a specific contract (`OUpgradeable` in our case) to mint and burn. `OUpgradeable` is an intermediate contract between LayerZero endpoint contract and `TetherTokenOFTExtension` token contract. When bridged to other chains, `TetherTokenOFTExtension` tokens are burned and when received from other chains, tokens are minted.

For this setup to work properly, `OAdapterUpgradeable` must be set as the peer of `OUpgradeable` on Ethereum mainnet. Similarly, `OUpgradeable` must be set as the peer of `AdapterUpgradeable` on Ink mainnet.

2.3 Trust Model & Assumptions

There are assumptions that are critical for system to work as intended.

1. `USDT.basisPointsRate` fee is assumed to be 0. `OAdapterUpgradeable._credit()` function relies on lossless transfers of tokens.
2. `USDT` is assumed not to be deprecated, as it might break other assumptions.
3. `OAdapterUpgradeable` is assumed not to be blacklisted in `USDT`, as it will break bridging functionality.
4. `OUpgradeable` is assumed to be set as `oftContract` in `TetherTokenOFTExtension`
5. LayerZero contracts and offchain infrastructure are assumed to be fully trusted and acting non-maliciously.
6. `OAdapterUpgradeable` is assumed to be deployed on mainnet.
7. `OUpgradeable` and `TetherTokenOFTExtension` are assumed to be deployed on Ink chain - EVM equivalent OP stack chain.

Certain roles within system have privilege access to critical functions and thus considered trusted:

1. Owner of `TetherTokenOFTExtension` contract is assumed to be trusted, as it might set malicious `oftContract`, blocklist and destroy funds.
2. Owner `OUpgradeable` is assumed to be trusted, as they can set malicious `peer` address.
3. Owner of `OAdapterUpgradeable` is assumed to be trusted, for the same reason as above.
4. `OApp` delegate of `OUpgradeable` is assumed to be trusted, as they can set malicious `SendLib` in LayerZero.
5. `OApp` delegate of `OAdapterUpgradeable` is assumed to be trusted, for the same reason as above.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Burning TetherTokenOFTExtension May Cause Balance Discrepancy Risk Accepted

5.1 Burning `TetherTokenOFTExtension` May Cause Balance Discrepancy

Design Low Version 1 Risk Accepted

CS-USDT0-001

All minted supply of `TetherTokenOFTExtension` is a bridged `USDT` token balance, locked on `OAdapterUpgradable` mainnet contract. Minting and burning using `lzReceive()` and `send()` functions of `OUpgradeable` contract maintains these balances in sync. However, `redeem()` and `destroyBlockedFunds()` functions of `TetherTokenOFTExtension` can be used to burn bridged `USDT` tokens without updating the locked balance on `OAdapterUpgradable` contract. This will cause a discrepancy between the total supply of `TetherTokenOFTExtension` and the locked balance on `OAdapterUpgradable` contract. Additionally, the `USDT` on mainnet does not have a way to burn some tokens of a specific address, which makes it impossible to update the locked balance on `OAdapterUpgradable` contract.

Risk accepted:

This is intended design decision.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0

High-Severity Findings	0

Medium-Severity Findings	0

Low-Severity Findings	1

- TetherTokenOFTExtension Has a Missing Gap Code Corrected

Informational Findings	1

- Updated Name and Symbol Do Not Affect Domain Separator Code Corrected

6.1 `TetherTokenOFTExtension` Has a Missing Gap

Design Low Version 1 Code Corrected

CS-USDT0-002

`TetherTokenOFTExtension` is intended to be an upgradeable contract. Base contracts such as `TetherTokenV2`, `TetherToken` and `EIP3009` rely on a `__gap` variable to reserve storage slots for future upgrades, as per storage layout convention. However, `TetherTokenOFTExtension` does not define such `__gap` variable, While this is not immediately an issue, in future upgrades, it may cause storage layout conflicts and complicate the upgrade process. Also `TetherToken` does not have gap.

If the contract is ever to be upgraded, storage layout compatibility check must be performed to ensure that the new contract does not overwrite existing storage slots.

Code corrected:

Gap was added to `TestTetherTokenOFTExtension` contract.

6.2 Updated Name and Symbol Do Not Affect Domain Separator

Informational Version 1 Code Corrected

CS-USDT0-003

During creation of `TestTetherTokenOFTExtension` contract, the name and symbol need to be provided in `initialize()` function. Also, `TetherTokenOFTExtension` has a `updateNameAndSymbol()` function, that allows owner to overwrite the name and symbol of the token. However it must be noted, that this overwrite does not affect `_domainSeparatorV4` that will always be calculated based on the original name and symbol provided during the initialization of the contract.

Code corrected:

Removed functionality entirely.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Rate Limiting Benefits

Note **Version 1**

Arbitrarily large amount of USDT can be bridged from one chain to another using USDT0. In event of breach, unrestricted asset movements can lead to significant market impact and de-peg the token value. Pattern such as RateLimiter can provide protection against that, while limiting usability for certain cases as a trade-off

7.2 USDT Decimals Does Not Match OFTAdapterUpgradeable Expected Type

Note **Version 1**

USDT returns uint8 as a decimal value, but `OFTAdapterUpgradeable` during construction queries `IERC20Metadata(_token).decimals()` that is expected to be uint256. However, due to ABI encoding behavior, this should not cause any issues, since uint8 will be padded to uint256 as per the ABI encoding rules:

> uint<M>: enc(X) is the big-endian encoding of X, padded on the higher-order (left) side with zero-bytes such that the length is 32 bytes.