GA GUARDIAN

USDT0

Tether OFT

Security Assessment

January 14th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, windhustler

Client Firm USDT0

Final Report Date January 14, 2025

Audit Summary

USDT0 engaged Guardian to review the security of their Tether OFT token on Ink. From the 2nd of January to the 7th of January, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Ink**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Fork Test Configuration Verification: https://github.com/GuardianAudits/usdt0-oft-contracts

Table of Contents

Project Overview

Project Summary

Project Name	USDT0
Language	Solidity
Codebase	https://github.com/Everdawn-Labs/usdt0-oft-contracts, https://github.com/Everdawn-Labs/usdt0-tether-contracts-hardhat
Commit(s)	Commit (initial) usdt0-oft-contracts: e6cffe572e9c92e9778c465c04cfae3526a06109 Commit (initial) usdt0-tether-contracts-hardhat: 7d173daf768fd692a915f395c29f9cdc034250e6 Commit (final) usdt0-oft-contracts: a53ae5822b71a091ab07decccbf4f3801965d31b Commit (final) usdt0-tether-contracts-hardhat: 102f11ccd644c624c7dafd9171f54e9e312fdb7c

Audit Summary

Delivery Date	January 13, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Configuration Validation

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	7	0	0	5	0	2
Info	7	0	0	4	0	3

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
L-01	Transfers To Tether Contract Block Messages	Validation	Low	Resolved
L-02	OFT Transfers Avoid Fees	Warning	Low	Acknowledged
L-03	EOA Signatures Unexpectedly Become Invalid	Unexpected Behavior	Low	Acknowledged
L-04	Incorrect Decimal Configuration Allowed	Best Practices	Low	Resolved
L-05	Burned OFTExtension Tokens Leaves USDT In Adapter	Unexpected Behavior	Low	Acknowledged
L-06	Msg.value Is Lost With lzReceive	Unexpected Behavior	Low	Acknowledged
L-07	USDT Pause Causes Cross-chain Messages To Fail	Unexpected Behavior	Low	Acknowledged
I-01	Unnecessary Imports	Best Practices	Info	Resolved
I-02	Missing disableInitializers	Best Practices	Info	Resolved
I-03	Memory Arguments Can Be Calldata	Optimization	Info	Acknowledged
I-04	Style Inconsistencies	Informational	Info	Resolved
I-05	ERC20PermitUpgradeable Draft Used	Best Practices	Info	Acknowledged
I-06	Enforced Message Execution Options Recommendations	Best Practices	Info	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
I-07	OFTExtension Minted Can Be Unbacked	Best Practices	Info	Acknowledged

L-01 | Transfers To Tether Contract Block Messages

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

The Tether token does not allow transfers or mints to the token address in the `_beforeTokenTransfer` function. However in the `OUpgradeable` contract there is no validation which prevents users from initiating cross-chain OFT transfers which would send tokens to the destination chain Tether token contract.

As a result any initiated OFT transfers with the to address as the Tether token on the destination chain will be stuck in the destination chain inbox and be un-executable, effectively trapping the users funds.

Recommendation

This may be resolved by storing the address of the Tether token contract on each corresponding `dstEid` in a mapping and validating that the `to` address is not the destination's Tether contract by overriding the `send` function.

Alternatively, the `_credit` function could include logic to send the bridged tokens to a trusted holding address if the `to` address is the Tether token contract.

If no code changes should be implemented for this issue, be sure to clearly document and warn users of this risk.

Resolution

USDT0 Team: Fixed with frontend validation.

L-02 | OFT Transfers Avoid Fees

Category	Severity	Location	Status
Warning	Low	OAdapterUpgradeable.sol	Acknowledged

Description

In the default `_debit` implementation for the `OFTAdapterUpgradeable` the `amountReceivedLD` is the same as the `amountSentLD` which is transferred from the user.

However if fees are enabled for the Tether token implementation at `0xdac17f958d2ee523a2206206994597c13d831ec7` on Ethereum then the amount received by the Adapter contract on Ethereum will be less than the `amountReceivedLD` which the user is credited with on the destination chain.

Firstly, this allows users to avoid the transfer tax on Ethereum. Secondly this can create balance underflow issues for users transferring their Tether back to Ethereum.

Consider the following scenario:
• The transfer tax is assigned as 1% on Ethereum
• User A uses OFT send to transfer 100 USDT from Ethereum to Arbitrum
• The Adapter contract receives 99 USDT and the 1 USDT fee is sent to the Tether owner address
• The Adapter contract balance is now 99 USDT
• User A receives 100 USDT on Arbitrum
• User A sends their 100 USDT back to Ethereum
• User A's message execution on Ethereum reverts and is stuck in the message channel because the Adapter contract only holds 99 USDT
• User A's 100 USDT are effectively lost until more USDT is added to the Adapter contract allowing them to execute the message

Recommendation

Consider overriding the `_debit` and `_credit` functions of the `OFTAdapterUpgradeable` to account for the Tether token transfer fees if they are enabled by using pre and post transfer balance checks. Otherwise ensure that the fees are never enabled for the USDT token on Ethereum.

Resolution

USDT0 Team: Acknowledged.

L-03 | EOA Signatures Unexpectedly Become Invalid

Category	Severity	Location	Status
Unexpected Behavior	Low	SignatureChecker.sol: 47	Acknowledged

Description

In the `isValidSignatureNow` function of the `SignatureChecker` library an `isContract` bytecode check is performed to determine whether a normal EOA ECDSA signature validation should occur with `ECRECOVER` or a ERC1271 contract signature validation should be performed.

However in the future, with the implementation of eip 7702 or eip 7377 it may be possible for EOAs to house bytecode, in which case the EOA ECDSA signature validation can no longer be performed for these EOAs.

It is likely that many EOAs which opt to house bytecode will not have functionality to support ERC1271 contract signatures and are therefore rendered unable to sign messages for the USDT token.

Furthermore, existing signatures that have been issued from EOAs and remain unused become unexpectedly invalid as soon as bytecode is written to them.

Recommendation

Consider always performing the ECDSA signature validation and opting to check the ERC1271 signature if the ECDSA signature verification fails, similarly to the `OpenZeppelin` `SignatureChecker` library.

Resolution

USDT0 Team: Acknowledged.

L-04 | Incorrect Decimal Configuration Allowed

Category	Severity	Location	Status
Best Practices	Low	OUpgradeable.sol: 23	Resolved

Description

In the `OUpgradeable` contract the constructor accepts a decimals parameter which determines the conversion from shared decimals to local chain decimals during the reception of tokens on the destination chain.

There is no validation which ensures that the configured token has matching decimals to the one provided.

If a misconfiguration were to occur during deployment then a Critical issue could arise where USDT amounts are delivered on the affected destination chain at orders of magnitude higher than the amount sent.

Recommendation

Consider querying the token decimals directly instead of specifying them in the constructor similar to how it is done in the `OFTAdapterUpgradeable` `LayerZero` contract with `IERC20Metadata(_token).decimals()`.

Resolution

USDT0 Team: The issue was resolved in commit 7dd2007.

L-05 | Burned OFTExtension Tokens Leaves USDT In Adapter

Category	Severity	Location	Status
Unexpected Behavior	Low	Global	Acknowledged

Description

The `OFTExtension` token has methods to burn token amounts outside of OFT transfers such as the `redeem` and `destroyBlockedFunds` `onlyOwner` functions.

When `OFTExtension` tokens are burned on remote chains, the corresponding USDT tokens which back these burned tokens remain in the `OAdapterUpgradeable` contract on Ethereum.

This behavior can be misleading as the `totalSupply` of USDT on Ethereum can misrepresent all of the USDT which is able to be bridged back to Ethereum.

Recommendation

If this behavior is not desired than be sure to burn the corresponding amount of USDT from the `OAdapterUpgradeable` contract on Ethereum when burning `OFTExtension` amounts on remote chains.

Otherwise be aware of this discrepancy when relying on the `totalSupply` of USDT or reading the balance of the `OAdapterUpgradeable` contract on Ethereum.

Resolution

USDT0 Team: Acknowledged.

L-06 | Msg.value Is Lost With lzReceive

Category	Severity	Location	Status
Unexpected Behavior	Low	OAdapterUpgradeable.sol	Acknowledged

Description

If users specify `msg.value` in the message execution options, this value will be passed by the executor while calling `lzReceive` on the destination chain. The value will be lost as it just ends up as a balance of `OUpgradeable` or `OAdapterUpgradeable` with no way of retrieving it.

Recommendation

The balance of `OUpgradeable` or `OAdapterUpgradeable` can be retrieved by adding an admin function to transfer the balance to the owner. Otherwise, make sure to document this behavior in the docs.

Resolution

USDT0 Team: Acknowledged.

L-07 | USDT Pause Causes Cross-chain Messages To Fail

Category	Severity	Location	Status
Unexpected Behavior	Low	Global	Acknowledged

Description

The USDT contract on Ethereum can pause transfers. If this happens, any in-flight messages will fail due to the inability to transfer USDT from the `OAdapterUpgradeable` contract.

Recommendation

If USDT is paused on Ethereum, cross-chain messages and bridged USDT should be paused on all the connected chains.

Resolution

USDT0 Team: Acknowledged.

I-01 | Unnecessary Imports

Category	Severity	Location	Status
Best Practices	Info	OUpgradeable.sol: 4	Resolved

Description

The `OUpgradeable` contract file includes several unnecessary imports from the `@layerzerolabs/oft-evm-upgradeable` package which are currently unused:

- `MessagingFee`
- `SendParam`
- `OFTReceipt`

Recommendation

Remove these unnecessary imports.

Resolution

USDT0 Team: The issue was resolved in commit 85787a7.

I-02 | Missing disableInitializers

Category	Severity	Location	Status
Best Practices	Info	Global	Resolved

Description

In the `OFTAdapterUpgradeable` and `OUpgradeable` contracts the constructors do not include calls to `_disableInitializers` to disable the `initializer` functions from being called on the implementation contract directly.

This is not a significant risk as the upgradeable contracts are not UUPS proxies, however it is a best practice to disable the initializers for upgradeable contracts.

Recommendation

Consider including a call to the `_disableInitializers` function in the constructor for the `OFTAdapterUpgradeable` and `OUpgradeable` contracts.

Resolution

USDT0 Team: The issue was resolved in commit 18c2b4d.

I-03 | Memory Arguments Can Be Calldata

Category	Severity	Location	Status
Optimization	Info	OFTExtension.sol: 50	Acknowledged

Description

The `updateNameAndSymbol` function accepts two `string memory` parameters which are never modified and only written to storage. These parameters do not need to be copied into memory and can instead remain as references to calldata.

Recommendation

Consider using the `calldata` location for the `_name` and `_string` parameters in the `updateNameAndSymbol` function.

Resolution

USDT0 Team: Acknowledged.

I-04 | Style Inconsistencies

Category	Severity	Location	Status
Informational	Info	OFTExtension.sol	Resolved

Description

Indentation of functions in the `TetherTokenOFTExtension` contract is inconsistent. Inconsistent parameter naming convention in `TetherTokenOFTExtension` contract: `mint` function uses `_destination` while `burn` uses `from` without underscore.

Recommendation

Fix the style inconsistencies by formatting the code properly and having a consistent naming convention.

Resolution

USDT0 Team: The issue was resolved in commit f241b88.

I-05 | ERC20PermitUpgradeable Draft Used

Category	Severity	Location	Status
Best Practices	Info	ERC20PermitUpgradeable.sol	Acknowledged

Description

The version of `ERC20PermitUpgradeable` used for the `TetherToken` contract is an outdated draft state version from `OpenZeppelin`.

Though no issues have been identified with this draft version of the `ERC20PermitUpgradeable` contract, the most up to date non-draft version of the `OpenZeppelin` `ERC20PermitUpgradeable` contract should ideally be used, accompanied by a Solidity version bump to a more recent version of Solidity.

Recommendation

Consider updating the version of the `OpenZeppelin` `ERC20PermitUpgradeable` contract that is used as well as the version of Solidity that is used.

Resolution

USDT0 Team: Acknowledged.

I-06 | Enforced Execution Options Recommendations

Category	Severity	Location	Status
Best Practices	Info	layerzero.config.ts.sol	Acknowledged

Description

LayerZero configuration files set the `gas` value for `ExecutorOptionType.LZ_RECEIVE` and `ExecutorOptionType.COMPOSE` options to 80k. `msgType` equal to `1` with option type `ExecutorOptionType.LZ_RECEIVE` is the case when `sendCompose` is not called on the destination chain and will have lower gas requirements than the `msgType` equal to `2` with option type `ExecutorOptionType.LZ_RECEIVE` which does call `sendCompose`.

The maximum message bytes size in the default LayerZero send library is `10_000` bytes. The worst case scenario is when `sendCompose` is called on the destination chain within the `lzReceive` function with the maximum composed message size -- which is `9980` bytes as each message encodes the `amount`, `msg.sender` and `receiver`. This will add significantly to the total gas due to manipulating bytes in memory and saving the messages in the LayerZero contracts for later execution.

With very long messages, the total gas requirements for `lzReceive` can be as high as ~230k gas. `msgType` equal to `2` with option type `ExecutorOptionType.COMPOSE` is the case when `lzCompose` is called on the destination chain in a separate transaction. Anyone is free to define their own contract that implements `lzCompose` and setting gas to `80k` might be too high if the logic inside `lzCompose` is very simple.

Recommendation

Benchmark the gas requirements for each option and message type on mainnet and enforce the gas limit to ensure there are no failed messages.
The 80k gas limit for `ExecutorOptionType.LZ_RECEIVE` with composed messages is insufficient, as testing showed longer messages can exceed this threshold. Consider increasing the limit while taking into account the trade-off between ensuring there are no pending messages and the increased gas costs for users that are sending short messages for compose.

For example, with 100 bytes of data for a composed message the `lzReceive` for the non-adapter OFT was measured at over 83,000 gas in testing. Considering that it may be common for users to include messages of around or over 100 bytes with composed messages, you might consider adjusting the enforced gas requirement for composed messages to be 100,000 units to be safest and cover longer composed messages by default.

I-07 | OFTExtension Minted Can Be Unbacked

Category	Severity	Location	Status
Best Practices	Info	OFTExtension.sol: 34	Acknowledged

Description

The system was reviewed with the assumption that `USDT` tokens get locked on Ethereum and are bridged to other chains, Ink and Berachain.

If we assume that bridging is allowed in all directions the system should work fine as every `USDT` on Ink/Berachain must have originated from Ethereum `USDT` being locked first.

If there is bridging between Ink/Berachain, tokens are burned on source and minted on destination.

As the admin of the `OFTExtension` contract can set any address to mint its tokens, there is a risk of minting tokens that are not backed by `USDT` on Ethereum.

Recommendation

Make sure that the `OFTExtension` contract only mints tokens that are backed by USDT on Ethereum.

Resolution

USDT0 Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits