PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Everdawn

10 January 2025

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Everdawn on the Ethereum and Ink network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Everdawn
URL	https://www.everdawn.to/
Platform	Ethereum, Ink
Language	Solidity
Preliminary Contracts	https://github.com/Everdawn-Labs/usdt0-tether-contracts-hardhat/commit/a99c8d9de92f30dc18c8d7a98c88b2e977ad06e2 https://github.com/Everdawn-Labs/usdt0-oft-contracts/commit/e6cffe572e9c92e9778c465c04cfae3526a06109
Resolution #1	https://github.com/Everdawn-Labs/usdt0-oft-contracts/commit/a53ae5822b71a091ab07decccbf4f3801965d31b
Resolution #2	

1.2 Contracts Assessed

Name	Contract	Live Code Match
OAdapterUpgradeable (ETH)	Proxy: 0x6C96dE32CEa08842dcc4058c14d3aaAD7Fa41dee Implementation: 0xCD979B10A55FCdAC23ec785CE3066c6ef8a479A4	MATCH
OUpgradeable (INK)	Proxy: 0x1cB6De532588fCA4a21B7209DE7C456AF8434A65 Implementation: 0x2257df4b93d2A55ED553194cAbEcD851A346FF89	MATCH
TetherToken	Dependency	MATCH
EIP3009	Dependency	MATCH
TetherTokenV2	Dependency	MATCH
WithBlockedList	Dependency	MATCH
ArbitrumExtension	Not deployed	UNDEPLOYED
CeloExtension	Not deployed	UNDEPLOYED
FeeCurrencyWrapper	Not deployed	UNDEPLOYED
OFTExtension	Proxy: 0x0200C29006150606B650577BBE7B6248F58470c1 Implementation: 0x4a5cd36c33D9F2986e5d39fF5B9918af547a6e0E	MATCH
WrapperExtension	Not deployed	UNDEPLOYED

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	1	-	-	1
Informational	11	1	2	8
Total	**12**	**1**	**2**	**9**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 OAdapterUpgradeable

ID	Severity	Summary	Status
01	INFO	Some implementation contracts can be initialized by anyone	✓ RESOLVED

1.3.2 OUpgradeable

ID	Severity	Summary	Status
02	LOW	Non-executable token transfers could be created through `OUpgradeable`	ACKNOWLEDGED
03	INFO	Insufficient validation	✓ RESOLVED
04	INFO	Typographical issues	PARTIAL

1.3.3 TetherToken

ID	Severity	Summary	Status
05	INFO	`WithBlockedList.sol` does not have storage gaps and `TetherToken.sol` has only one	ACKNOWLEDGED
06	INFO	Typographical issues	ACKNOWLEDGED

1.3.4 EIP3009

No issues found.

1.3.5 TetherTokenV2

ID	Severity	Summary	Status
07	INFO	Typographical issues	ACKNOWLEDGED

1.3.6 WithBlockedList

ID	Severity	Summary	Status
08	INFO	Typographical issues	ACKNOWLEDGED

1.3.7 ArbitrumExtension

No issues found.

1.3.8 CeloExtension

No issues found.

1.3.9 FeeCurrencyWrapper

No issues found.

1.3.10 OFTExtension

ID	Severity	Summary	Status
09	INFO	Mint function is not callable by the owner	ACKNOWLEDGED
10	INFO	Typographical issues	PARTIAL

1.3.11 WrapperExtension

ID	Severity	Summary	Status
11	INFO	Insufficient validation	ACKNOWLEDGED
12	INFO	Typographical issues	ACKNOWLEDGED

2 Findings

2.1 OAdapterUpgradeable

`OAdapterUpgradeable` is an implementation contract that introduces the OFT logic to an ERC20 token via the lock/unlock mechanism on the source chain. Only one adapter type OFT must be deployed to guarantee finality for token transfers.

2.1.1 Privileged Functions

- `setMsgInspector [OWNER]`
- `setEnforcedOptions [OWNER]`
- `setPreCrime [OWNER]`
- `setPeer [OWNER]`
- `setDelegate [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.1.2 Issues & Recommendations

Issue #01	Some implementation contracts can be initialized by anyone
Severity	INFORMATIONAL
Description	Within the proxy pattern, a contract's implementation is initialized via a function that has an initializer modifier. This pattern is inherited from the Initializable contract. To avoid the initialization of the implementation by unauthorised actors, a _disableInitializers call is required in any constructor of the main contract. In transparent upgradable proxies, this does not pose a risk but it is still considered a best practice.
Recommendation	Consider adding the `_disableInitializers` call in the constructor of `OUpgradeable` and `OAdapterUpgradeable`.
Resolution	RESOLVED

2.2 OUpgradeable

`OUpgradeable` is an implementation contract that is almost identical to `OAdapterUpgradeable` but with the difference of being privileged by the `TetherTokenOFTExtension` token to burn/mint quantities upon sending/receiving cross chain transfers.

2.2.1 Privileged Functions

- `setMsgInspector [OWNER]`
- `setEnforcedOptions [OWNER]`
- `setPreCrime [OWNER]`
- `setPeer [OWNER]`
- `setDelegate [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.2.2 Issues & Recommendations

Issue #02	**Non-executable token transfers could be created through `OUpgradeable`**
Severity	LOW SEVERITY

Description	`OUpgradeable` is meant to be used as an enhanced version of the LayerZero OFT token. The main difference is that a separate token contract (`token_`) is used instead of acting as the ERC20 contract itself. The underlying token used is the `TetherTokenOFTExtension` contract, which inherits from the Tether contract.
	
	The core Tether contract blocks transfers if the `to` address is `address(this)`, basically blocking transfers to itself.
	
	`function _beforeTokenTransfer() {`
	`  [...]`
	`  require(`
	`  to != address(this),`
	`  "TetherToken: transfer to the contract address"`
	`  );`
	`  [...]`
	`}`
	
	Users are able to make a cross-chain token transfer to the address of the Tether token that will always revert when attempted to be executed on the destination chain because of Tether's check inside `_beforeTokenTransfer`.

Recommendation	Consider overriding `send()` inside `OUpgradeable` to add a validation that the `to` address is not the peer contract on the other side. There are two possible fixes. The first is to introduce a check in `OUpgradeable.send()` and revert if the `to` address is the address of the Tether token on the destination chain. This way, such transactions will revert on the source chain. The second option is to redirect such transfers to the `0xdead` address on the destination chain OFT as it is currently done for transfers to `address(0)`. In this scenario, the transaction on the destination chain will not revert.

Resolution	ACKNOWLEDGED The team commented that it is not an issue and will be handled in the front end.

Issue #03	**Insufficient validation**
Severity	INFORMATIONAL
Description	Inside the constructor of `OUpgradeable`, a check that the provided decimals are equal to the ones of the underlying token is missing. This is necessary since the contract is not the token itself (which is the case for standard OFTs) but it uses another token contract under the hood, and the decimals between the two contracts must be identical to ensure proper `decimalConversionRate`.
Recommendation	Consider adding the check.
Resolution	RESOLVED

Issue #04	**Typographical issues**
Severity	INFORMATIONAL
Description	The `MessagingFee`, `SendParam`, and `OFTReceipt` imports are not used and can be removed. — `token()` should be moved below the `constructor()` —it is considered a convention to keep the constructor at the top and place all functions after that.
Recommendation	Consider fixing the typographical issues
Resolution	PARTIALLY RESOLVED

2.3 TetherToken

TetherToken is version one of USDT.

2.3.1 Privileged Functions

- `mint [OWNER]`
- `redeem [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.3.2 Issues & Recommendations

Issue #05	`WithBlockedList.sol` does not have storage gaps and `TetherToken.sol` has only one
Severity	INFORMATIONAL
Description	TetherToken is inherited by `TetherTokenV2` and `ArbitrumExtension.sol`. Both have storage variables right after the storage slots of `TetherToken`, which means that in order to upgrade `WithBlockedList` or `TetherToken`, there is only one extra storage slot to work with otherwise there is the risk of overwriting the storage slots of child contracts. There is also a way to upgrade the contract by attaching a new child contract that would override existing functions and introduce new storage variables if these functions are not supposed to interact with existing state stored in private variables.
Recommendation	Consider implementing proper storage gaps in `WithBlockedList` and `TetherToken`.
Resolution	ACKNOWLEDGED

Issue #06	Typographical issues
Severity	INFORMATIONAL
Description	Consider adding natspec comments to undocumented functions for better readability.
Recommendation	Consider fixing the typographical issues.
Resolution	ACKNOWLEDGED

2.4 EIP3009

`EIP3009` is an abstract contract that implements EIP-3009.

EIP-3009, also known as "Transfer With Authorization," is an Ethereum Improvement Proposal that introduces a contract interface for transferring fungible assets via signed authorizations. The proposal aims to enable meta-transactions and atomic interactions with ERC-20 token contracts.

2.4.1 Issues & Recommendations

No issues found.

2.5 TetherTokenV2

`TetherTokenV2` is an extension of the original `TetherToken` that adds the permit functionality and EIP-3009.

2.5.1 Privileged Functions

- `mint [OWNER]`
- `redeem [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.5.2 Issues & Recommendations

Issue #07	Typographical issues
Severity	INFORMATIONAL
Description	The Solidity version should come right after the license identifier.
Recommendation	Consider fixing the typographical issues
Resolution	ACKNOWLEDGED

2.6 WithBlockedList

`WithBlockedList` is an extension used in `TetherToken` that provides a blocklist functionality.

2.6.1 Privileged Functions

- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.6.2 Issues & Recommendations

Issue #08	Typographical issues
Severity	INFORMATIONAL
Description	Consider adding natspec comments to undocumented functions for better readability.
Recommendation	Consider fixing the typographical issues
Resolution	ACKNOWLEDGED

2.7 ArbitrumExtension

`ArbitrumExtension` wraps the Tether token on the Arbitrum chain to provide the native bridge with the ability to mint/burn USDT upon sending/receiving cross chain transfers.

2.7.1 Privileged Functions

- `mint [OWNER]`
- `redeem [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`
- `bridgeMint [only Gateway]`
- `bridgeBurn [only Gateway]`

2.7.2 Issues & Recommendations

No issues found.

2.8 CeloExtension

`CeloExtension` wraps the Tether contract on the Celo chain to allow users to pay for transactions with USDT.

2.8.1 Privileged Functions

- `creditGasFees [only VM]`
- `debitGasFees [only VM]`
- `mint [OWNER]`
- `redeem [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `setFeeCurrencyWrapper [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.8.2 Issues & Recommendations

No issues found.

2.9 FeeCurrencyWrapper

`FeeCurrencyWrapper` is used by the Celo VM instead of `CeloExtension` directly due to USDT having 6 decimals instead of 18.

2.9.1 Privileged Functions

- `creditGasFees [only VM]`
- `debitGasFees [only VM]`
- `mint [OWNER]`
- `redeem [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `setFeeCurrencyWrapper [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.9.2 Issues & Recommendations

No issues found.

2.10 OFTExtension

`OFTExtension` wraps the Tether token to give mint and burn privileges to the OFT contract.

2.10.1 Privileged Functions

- `mint [only OFT]`
- `burn [only OFT]`
- `redeem [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `setFeeCurrencyWrapper [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`

2.10.2 Issues & Recommendations

Issue #09	Mint function is not callable by the owner
Severity	INFORMATIONAL
Description	The mint function in `TetherTokenOFTExtension.sol` overrides the initial mint function that is callable by the owner. With this, only the OFT can mint and the owner cannot. It might be intended for the OFT to be able to mint on certain chains by first locking the original token in the adapter on Mainnet. Here is a case where this might be problematic: Attacker steals 1M of USDT, USDT owner attempts to block him to burn all of his tokens via `destroyBlockedFunds()` but the attacker manages to bridge the 1M USDT to chain X via LayerZero before this. On chain X, the attacker receives `TetherTokenOFTExtension` and there, the owner manages to block him and burn his tokens. Now on mainnet, the 1M are locked inside the OFT adapter and in order to be burned, the owner will have to block the OFT adapter, burn its tokens and then mint new ones minus this 1M. If the owner could instead mint tokens on chain X, he would be able to mint this 1M, bridge them back to himself and then burn it via `redeem()` or distribute it to the party they were stolen from without needing to block the adapter temporarily. Still, minting on chain X is risky if done for any other reason by the owner because if bridged back to the adapter, this might result in the adapter not having enough funds at some point and cause DOS.
Recommendation	Consider allowing the owner of the contract to have the ability to mint tokens for such edge cases.
Resolution	ACKNOWLEDGED

Issue #10	Typographical issues
Severity	INFORMATIONAL
Description	In `TetherTokenOFTExtension`, consider changing the `onlyAuthorizedSender` modifier error from "Only VM can call" to "Only OFT can call" — The code indentation on the OFTExtension contract is missing. — Consider adding Natspec comments to undocumented functions for better readability.
Recommendation	Consider fixing the typographical issues.
Resolution	PARTIALLY RESOLVED

2.11 WrapperExtension

`WrapperExtension` handles the migration from Tether v1 to Tether v2.

2.11.1 Privileged Functions

- `mint [OWNER]`
- `redeem [OWNER]`
- `destroyBlockedFunds [OWNER]`
- `addToBlockedList [OWNER]`
- `removeFromBlockedList [OWNER]`
- `renounceOwnership [OWNER]`
- `transferOwnership [OWNER]`
- `withdrawDifferentToken [OWNER]`
- `withdrawBalanceDifference [OWNER]`
- `setWithdrawable [OWNER]`

2.11.2 Issues & Recommendations

Issue #11	Insufficient validation
Severity	INFORMATIONAL
Description	Within the constructor, there should be a check that `_originalToken` is not `address(0)`.
Recommendation	Consider implementing the check.
Resolution	ACKNOWLEDGED

Issue #12	Typographical issues
Severity	INFORMATIONAL
Description	The `originalToken` variable could be formatted in uppercase, since it is a convention for constant variables. — Consider adding NatSpec comments to all the functions.
Recommendation	Consider fixing the typographical issues.
Resolution	ACKNOWLEDGED

PALADIN
BLOCKCHAIN SECURITY