Audit Report

Celer Flow Contracts

v1.0

September 20, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by FLOW Digital to perform a security audit of Celer Flow Contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.

2. Determine possible vulnerabilities, which could be exploited by an attacker.

3. Determine smart contract bugs, which might lead to unexpected behavior.

4. Analyze whether best practices have been applied during development.

5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/alilloig/celer-contracts
Commit	`02d2f6e7c4e11adfeded8b0d3a2cf9078d17d065`
Scope	All contracts were in scope.
Fixes verified at commit	`017cd433d94762fd0bc658053f11e2688adfb022` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Celer cBridge contracts allow projects built on Flow to bridge into other cBridge-supported chains, creating seamless multi-chain interoperability.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Low-Medium**	-
Level of documentation	**Medium**	-
Test coverage	**Low**	There are no test cases in the codebase.

Summary of Findings

No	Description	Severity	Status
1	Missing entitlement causes deposit transactions to fail	Minor	Resolved
2	Missing receiver capability validation leads to locked funds at Flow chain	Minor	Resolved
3	Verifying signatures may fail due to the execution limit	Informational	Acknowledged
4	Inconsistent return types cause obstacles in writing generic transactions and scripts	Informational	Resolved
5	Unused entitlement	Informational	Resolved
6	Getter functions are not marked with the view keyword	Informational	Resolved
7	Suboptimal balance update	Informational	Resolved

Detailed Findings

1. Missing entitlement causes deposit transactions to fail

Severity: Minor

In `transactions/safebox_deposit.cdc:13`, the `from` argument is not implemented as an authorized reference with the `FungibleToken.Withdraw` entitlement. This is required as the `deposit` function in `cadence/contracts/SafeBox.cdc:134` requires the reference type as `auth(FungibleToken.Withdraw)&{FungibleToken.Provider}`.

Consequently, the transaction will fail to execute.

Recommendation

We recommend implementing the correct entitlement.

Status: Resolved

2. Missing receiver capability validation leads to locked funds at Flow chain

Severity: Minor

A `FungibleToken.Receiver` capability named `receiverCap` is retrieved from a configured token public path in `contracts/SafeBox.cdc:186` for the receiver address.

However, no validation ensures the fetched capability supports the intended receiving token type. This oversight may cause the `executeDelayXfer` function to fail in `contracts/DelayedTransfer.cdc:101` if the receiver capability reference (indicated as `recRef`) does not align with the receiving token vault type.

Consequently, users may be unable to withdraw funds from Flow chain despite the corresponding burn transaction already executed on the counterparty chain.

We classify this issue as minor because the likelihood of encountering an invalid receiving capability at the specified public path is low. Potential causes include users holding a different token at the same path or interacting with malicious dApps that manipulate user capability paths.

A recovery plan could be initiated in which users must manually replace the incorrect capability with the correct capability in the configured path and re-execute the `executeDelayXfer` function. Nevertheless, this leads to operational overhead and decreased user experience.

Recommendation

We recommend borrowing `receiverCap` and calling the `isSupportedVaultType` function to ensure the receiver capability supports the receiving token vault type. Alternatively, we recommend returning an error earlier in the transaction instead of letting the transfer fail in `contracts/DelayedTransfer.cdc:101`.

Status: Resolved

3. Verifying signatures may fail due to the execution limit

Severity: Informational

In `contracts/cBridge.cdc:57`, the `verify` function accepts a variable-length array of signatures in the `sigs` argument. The function logic implements an inner loop inside an outer loop to find the matched signature, as seen in lines `61-87`.

This may cause the execution effort of such logic to reach the execution limit on Flow chain, causing the `verify` function to fail if there are many signatures.

Recommendation

We recommend implementing a dictionary for the `sigs` argument like `{signerPubKey: sig}` to avoid excessive iterations.

Status: Acknowledged

4. Inconsistent return types cause obstacles in writing generic transactions and scripts

Severity: Informational

Throughout the implementation of the fungible token contracts, the following functions do not return the `@{FungibleToken.Vault}` type value:

- `withdraw`
    - `contracts/ceAVAX.cdc:105`
    - `contracts/ceBNB.cdc:107`
    - `contracts/ceMATIC.cdc:84`
- `mintTokens`
    - `contracts/ceAVAX.cdc:199`
    - `contracts/ceBNB.cdc:201`
    - `contracts/ceDAI.cdc:178`
    - `contracts/ceMATIC.cdc:178`
    - `contracts/ceUSDT.cdc:199`
    - `contracts/ceWETH.cdc:199`

- `createEmptyVault`
  - `contracts/ceBNB.cdc:138`
  - `contracts/ceBNB.cdc:162`
  - `contracts/ceBUSD.cdc:139`
  - `contracts/ceDAI.cdc:115`

Consequently, the code complexity and maintainability will increase because each contract requires its own customized transaction and script files to return the correct resource types.

Recommendation

We recommend setting the return type as `@{FungibleToken.Vault}` so a generic transaction and script file can be used directly.

Status: Resolved

5. Unused entitlement

Severity: Informational

In `cadence/contracts/VolumeControl.cdc:3`, the `Update` entitlement is defined but not implemented, which reduces code readability and maintainability.

Recommendation

We recommend removing the entitlement or implementing it according to the business logic.

Status: Resolved

6. Getter functions are not marked with the view keyword

Severity: Informational

In Cadence 1.0, view functions are introduced to enforce getter functions that do not modify any state. However, this is not enforced in the following functions:

- `getSigners` in `cadence/contracts/cBridge.cdc`
- `delayTransferExist` and `getDelayBlockTs` in `cadence/contracts/DelayedTransfer.cdc`
- `hasMore`, `toUint64`, `toUint256`, `toAddress`, `toUFix64`, and `toString` in `cadence/contracts/Pb.cdc`
- `eqToken` in `cadence/contracts/PbPegged.cdc`
- `getTokenConfig` and `recordExist` in `cadence/contracts/PegBridge.cdc`
- `getTokenConfig` and `recordExist` in `cadence/contracts/SafeBox.cdc`

- `getEpochVolume` and `getLastOpTimestamp` in `cadence/contracts/VolumeControl.cdc`

Recommendation

We recommend applying the view keyword for the above-mentioned functions to ensure they do not mutate state.

Status: Resolved

7. Suboptimal balance update

Severity: Informational

In `cadence/contracts/RLY.cdc:133`, the `burnCallback` function sets the balance to zero regardless of the current balance. This is suboptimal as the balance may already be zero, making it a redundant operation.

This issue also affects the following contracts:

- `cadence/contracts/ceAVAX.cdc:133`
- `cadence/contracts/ceBNB.cdc:135`
- `cadence/contracts/ceBUSD.cdc:112`
- `cadence/contracts/ceDAI.cdc:112`
- `cadence/contracts/ceFTM.cdc:112`
- `cadence/contracts/ceMATIC.cdc:112`
- `cadence/contracts/ceUSDT.cdc:133`
- `cadence/contracts/ceWBTC.cdc:133`
- `cadence/contracts/ceWETH.cdc:133`

Recommendation

We recommend only setting the balance to zero if the balance is larger than zero. This can be achieved by placing the code into the if statement inside `cadence/contracts/RLY.cdc:130-132`.

Status: Resolved