Audit Report

Celer cBridge Flow

v1.0

May 14, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Celer Network to perform a security audit of Celer network cBridge smart 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 GitHub repository:

https://github.com/celer-network/cbridge-flow

Commit hash: `e8451a3681f1cc1260f26bf3567eb01f22abcbb8`

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

The submitted code implements Celer cBridge contracts supporting the Flow ecosystem. Projects built on Flow will be able 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.

Summary of Findings

No	Description	Severity	Status
1	Unbounded dictionary records might cause denial of service	Minor	Acknowledged
2	Adding delay transfer should not be allowed when the contract is paused	Minor	Acknowledged
3	Public keys are not validated when updating or resetting signers	Minor	Acknowledged
4	Duplicate public keys are not removed	Minor	Acknowledged
5	Missing logic validations during struct initialization might cause temporary denial of service	Minor	Acknowledged
6	Sensitive resources can easily be shared and not revoked	Minor	Acknowledged
7	Misconfiguration of chain identifier values might lead to replay attack possibility	Minor	Acknowledged
8	Token receiving capability may not exist, which leads to failure of deposits	Minor	Acknowledged
9	Delay threshold check should use greater than or equal symbol	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	Code is easy to read and understand, although there are some places where code complexity can be reduced.
Level of documentation	Medium-High	Certain parts of the codebase are well documented while other parts, such as decoding algorithms for raw data, could benefit from more low-level documentation.
Test coverage	Medium-High	-

Detailed Findings

1. Unbounded dictionary records might cause denial of service

Severity: Minor

In `contracts/PegBridge.cdc:86`, the `records` dictionary is used to record existing minting and burning hash identifiers as seen in lines `223` and `263`. While recording the identifiers on-chain are required to prevent replay attacks, this would cause the account's storage capacity to increase limitlessly. If the account doesn't hold enough Flow tokens proportional to the increased storage capacity, this would cause new transactions to fail as it would put the account over it's storage capacity.

This issue is also found in `contracts/SafeBox.cdc` in lines `149` and `185` where it holds the deposit and withdrawal hash identifiers.

Recommendation

We recommend modifying the implementation to use a nonce-based signing system similar to the ERC20 permit standard. This would eliminate the use of tracking the hash identifiers on-chain, yet still prevent replay attacks.

Status: Acknowledged

The Celer team states that their system design requires on-chain records to avoid replay attacks. They are well aware of the FLOW token cost for storage and priced this into their operating cost.

The risk of a DoS attack is minimal due to several reasons:

- The team is able to increase the token's minimum send requirements to increase the attacker's cost
- Existing safeguard measures are implemented such as `DelayedTransfer` and `VolumeControl`, which slow down fund release
- At current storage pricing, adding 1 FLOW token allows 100MB extra storage
- Dynamic transaction fee will be enabled soon

Hence, practically an attacker is unlikely to gain anything by spamming cross-chain transfers.

2. Adding delay transfer should not be allowed when the contract is paused

Severity: Minor

In `contracts/DelayedTransfer.cdc:73`, adding delay transfers through the `executeDelayXfer` function is allowed even if the contract is paused. If a catastrophic

event happens, users may continuously get exploited since new delay transfers can still be added while most other functionalities in the contract are inaccessible.

Recommendation

We recommend preventing delayed transfers if the contract is paused.

Status: Acknowledged

The Celer team states that the `addDelayXfer` function is an account access scope and only `PegBridge` and `SafeBox` contracts will call it, in which both contracts' `isPaused` value is verified first when executing the function, hence this isn't an issue. Fundamentally by design, delay transfer is only used to control assets leaving the system. They agree a better name could be delayWithdraw or delayFunding but they decided to keep naming consistent with Solidity contracts.

3. Public keys are not validated when updating or resetting signers

Severity: Minor

In `contracts/cBridge.cdc:36`, the passed signer's public keys are directly set without verifying whether the public keys are valid and conform to the hardcoded signature algorithm which is `ECDSA_secp256k1`. If an invalid public key or a public key with a different signature algorithm is passed, the public key construction in lines `61` to `64` would fail, potentially causing the whole cBridge contract to be unusable. This issue is also present in the `updateSigners` function in lines `157-172`.

We consider this to be a minor issue since it can only be caused by the admin or existing signers.

Recommendation

We recommend validating the public keys by constructing the `PublicKey` structure to make sure they are valid `ECDSA_secp256k1` public keys in both the `resetSigners` and the `updateSigners` functions.

Status: Acknowledged

The Celer team states that they don't expect this to be an issue due to several reasons:

- No security implication as invalid keys will cause the `verify` function to fail
- The team's Golang tools will verify the public key to be valid before setting it
- Admin can set correct public key to recover if a wrong key is set by accident

4. Duplicate public keys are not removed

Severity: Minor

In `contracts/cBridge.cdc:36`, the passed signer's public keys are directly set without verifying whether they are duplicates or not. If a public key is twice in the `signers` dictionary, the signature validation can be replayed and potentially allow the signer to hold more signing power than intended. This issue is also present in the `updateSigners` function from in lines `157-172`.

We consider this to be a minor issue since it can only be caused by the admin or existing signers.

Recommendation

We recommend deduplicating public keys in both the `resetSigners` and the `updateSigners` functions.

Status: Acknowledged

The Celer team states that new signers are not arbitrary data but are first validated by Celer SGN system, so duplication isn't a concern. They agree it's best to have additional enforcement inside the contract and plan to add `toLower` for admin's `resetSigners` function.

5. Missing logic validations during struct initialization might cause temporary denial of service

Severity: Minor

In `contracts/PegBridge.cdc:52-53`, the `TokenCfg` struct's `minBurn` and `maxBurn` values are not validated. Specifically, the minimum burn amount should be validated to be lower than the maximum burn amount while the maximum burn amount should be validated to be higher than the minimum burn amount if the value is not 0. If the values are initialized incorrectly, the burn function would fail in lines `255-258`. This issue is also found present in `contracts/SafeBox.cdc` from lines `49-50`.

We consider this to be a minor issue since it can only be caused by the admin.

Recommendation

We recommend performing validation as mentioned above in `contracts/PegBridge.cdc:52-53` and `contracts/SafeBox.cdc:49-50` respectively.

Status: Acknowledged

The Celer team states that there's no plan to add min is equal to or greater than max check in the contract as it's already checked in their tool.

6. Sensitive resources can easily be shared and not revoked

Severity: Minor

Throughout the codebase, every admin resource has a `createXAdmin` method (where `X` is a placeholder for the resource) that allows the existing resource to create a new admin resource that can be assigned to another account (e.g. in `contracts/SafeBox.cdc:114-116`). Due to this, admin resource ownership can be provided to multiple accounts, which increases the attack surface. Admin permissions cannot be revoked in the current implementation, which exacerbates the issue.

Recommendation

We recommend removing the `createXAdmin` method from all the admin resources and implementing the private capability feature to allow revocation of permissions.

Status: Acknowledged

The Celer team states that creating a new admin resource is needed to support their product requirement that later transfers ownership/admin to a DAO contract.

7. Misconfiguration of chain identifier values might lead to replay attack possibility

Severity: Minor

In `contracts/PegBridge.cdc:190` and `contracts/SafeBox.cdc:122`, the chain identifier values use the `chID` argument, which is supplied by the contract instantiator. That argument is concatenated with the contract's address and name in the next line. Since the chain identifier is solely based on what value the contract instantiator provides, this will open up a replay attack possibility if the same chain identifier is used across multiple deployments, for example on mainnet and testnet. As a result, this would cause the `domainPrefix` value to be the same across the mainnet and testnet contracts (assuming the same contract address and name), potentially allowing an attacker to replay mint and withdrawal operations in `contracts/PegBridge.cdc:213-214` and `contracts/SafeBox.cdc:172-173`.

We consider this issue to be minor since it would only occur if the contract instantiator instantiated the chain identifiers incorrectly.

Recommendation

We recommend clearly documenting the necessity to use unique chain ID arguments across different deployments.

Status: Acknowledged

The Celer team states that since Flow has no native chain identifier support, they defined different chain identifier values for Flow emulator, testnet, and mainnet environment. Their

contract-deploy tool guarantees that each contract's chain identifier value is correctly set. An alternative method would be to hardcode the chain identifier in the contract file and manually keep it different for each chain, which is a less preferable method.

8. Token receiving capability may not exist, which leads to failure of deposits

Severity: Minor

In `contracts/SafeBox.cdc:151`, the capability for receiving funds gets accessed. There is no validation though to ensure `self.account` has the public capability stored at `tokenCfg.vaultPub`. As a result, the `deposit` operation would fail every time for the specific whitelisted token.

Recommendation

We recommend creating the public fungible token receiving capability if the capability doesn't exist during the `addTok` function execution in line `95`.

Status: Acknowledged

The Celer team states that adding `vaultPub` is handled by their tool, hence it is not needed to be added to the contract.

9. Delay threshold check should use greater than or equal symbol

Severity: Informational

In `contracts/PegBridge.cdc:229` and `contracts/SafeBox.cdc:191`, the if statement check uses the greater than symbol > to determine whether the mint or withdrawal amount is required to be delayed. As any amount that reaches the delay threshold should be delayed, the check should include equality of the amounts.

Recommendation

We recommend replacing the greater than symbol > with the greater than or equal symbol >=.

Status: Acknowledged

The Celer team states that the comparison is needed to be consistent with their Solidity contracts and product requirements.