BOB

On-ramp

Smart Contracts Audit

Apr 22, 2024

Common Prefix

Overview

Introduction

Common Prefix was commissioned to perform a security audit on BOB's bob-onramp repository at commit hash 297eeb1faeb7387bab142b4d2e0bcc6ef191747a.

The files inspected are the following:

`OnRamp.sol`
`OnRampFactory.sol`

Protocol Description

BOB's Onramp contracts facilitate users' onboarding onto the BOB L2 network without the need to hold any Ethereum assets beforehand. They enable users to exchange their BTC for ERC20 tokens pegged to BTC, such as tBTC and wBTC. These tokens are provided by Liquidity Providers (LPs).

BTC holders initiate the onboarding process by transferring a BTC amount to a BTC address corresponding to the LP of an Onramp contract. After this transaction, a trusted relayer posts to the Onramp contract a proof for the transaction and subsequently transfers the corresponding token amount (minus the fees) to a specified BOB address provided by the user.

Each LP has its own Onramp contract, which should be created through the Onramp Factory contract. Each Onramp contract is designed to hold only one type of tokens, provided by its respective LP. Additionally, the LP contributes a small amount of ETH, which is distributed to each user exchanging their BTC. These ETH are intended to cover the gas fees for the user's initial transactions on the BOB network.

LPs earn fees in the form of a percentage, determined by them, of the exchanged amount. The LP has the ability to adjust the fee percentage or collect their deposits and fees. However, any

changes to the fee percentage or collection of funds require a certain delay (6 hours) to pass, allowing the relayer sufficient time to execute all pending transactions before the changes take effect. This delay mechanism ensures that users are informed about any modifications to fees or fund collection policies before they occur.

Disclaimer

Note that this audit does not give any warranties on the bug-free status of the given smart contracts, i.e. the evaluation result does not guarantee the nonexistence of any further findings of security issues. This audit report is intended to be used for discussion purposes only. Functional correctness should not rely on human inspection but be verified through thorough testing. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the project.

Findings Severity Breakdown

The findings are classified under the following severity categories according to the impact and the likelihood of an attack.

Level	Description
Critical	Logical errors or implementation bugs that are easily exploited and maylead to any kind of loss of funds
High	Logical errors or implementation bugs that are likely to be exploited and may have disadvantageous economic impact or contract failure
Medium	Issues that may break the intended contract logic or lead to DoS attacks
Low	Issues harder to exploit (exploitable with low probability), issues that lead to poor contract performance, clumsy logic or seriously error-prone implementation
Informational	Advisory comments and recommendations that could help make the codebase clearer, more readable and easier to maintain

Findings

Critical

No critical issues found.

High

No high issues found.

Medium

MEDIUM-1	`executeSwap` is allowed to be called unconditionally by the relayer
Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

In the OnRamp.sol contract the following requirement is essentially a no-op:

```
        require(
            updateStart <= block.timestamp + UPDATE_DELAY,
            "Not allowed to execute"
        );
```

Since either the updateStart has not been set by the owner, and therefore is zero, or it has been set and equals the timestamp of a past block. In both cases the condition above is trivially satisfied and therefore puts no restrictions.

Recommendation

We suggest fixing the `require()` condition so that the `executeSwap` function is allowed to be executed only when either the `updateStart` parameter has not been set or, if it has been set, the `block.timestamp` lies within the `(updateStart, updateStart + UPDATE_DELAY)` time window.

Alleviation

The team fixed the issue at commit hash 416df89fffbe34b5abdb984c07dca3de59f3d106.

Low

LOW-1	No `deposit` function exists and the `receive` function does not check `msg.sender`
Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

In the `OnRamp.sol` contract the funds of the contract are supposed to be sent by the owner of the contract via an immediate transfer. This practice is error prone as it does not protect the liquidity providers from sending funds to incorrect onRamp contracts, which cannot be claimed or returned.

Similarly, the receive function does not check that the msg.sender is the contract's owner, allowing mistaken transfers.

Recommendation

We suggest implementing a `deposit` function which is allowed to be called only by the contract's owner and clearly documenting that liquidity providers should use this function to transfer funds to an onRamp contract for their own safety. In addition, consider reverting the execution of the `receive` function when `msg.sender` is not the contract's owner.

Alleviation

The team fixed the issue at commit hash d09facee46f8791390981199ed38331e4eed25c9.

LOW-2	`executeSwap` could fail due to insufficient gas
Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

The `executeSwap` function in the OnRamp contract uses the EVM `transfer` method to send ETH to the specified `receiver` address. However, this method forwards only a fixed amount of 2300 gas, which may not be enough for more complex operations in the receiving contract. As a consequence, `executeSwap` will always fail for such a receiver, essentially concluding to a DoS.

Recommendation

We suggest using a low level `call` to transfer ETH funds instead and checking that it was successful.

Alleviation

The team fixed the issue at commit hash [1b7276d672eee451f212c8f9dc46f3b5d2708ee9](1b7276d672eee451f212c8f9dc46f3b5d2708ee9).

LOW-3	Lack of validation of the onramp contracts
Contract(s)	`OnrampFactory.sol`
Status	**Resolved**

Description

While the `OnrampFactory` contract maintains an array containing all the onramp contracts created through its `createOnramp` function, this array serves no purpose within the protocol. Specifically, in the `proveBtcTransfer` function, which is called by the relayer, the relayer supplies the onramp contract related to the transaction. However, there's no validation to ensure that this contract is listed among those held by the factory.

Recommendation

We suggest implementing a mapping to indicate whether an onramp contract was created through the factory contract. This mapping can then be utilized to validate the onramp contract provided by the relayer in the `proveBtcTransfer` function.

Alleviation

The team fixed the issue at commit hash [36a18bd37f1d7a802d4d97690592fa676dbdaf18]

LOW-4	`feeDivisor` and `gratuity` parameters can be set to arbitrary values
Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

The onramp owner currently has the ability to set the `feeDivisor` arbitrarily low, potentially enforcing excessively high fees. Additionally, there is no lower bound on the amount of gratuity, which is the small amount of ETH offered by the onramp owners to the users for gas.

Recommendation

We suggest setting constant lower and upper limit values for each of the `feeDivisor` and `gratuity` parameters to ensure that only a small portion of the exchanged value is withheld as compensation by the liquidity provider but also that the receiver is eligible to a non-zero amount of `gratuity`.

Alleviation

The team fixed the issue at commit hash 88a4856eb6d0d33429e3f06629704adaff882960.

Informational/Suggestions

INFO-1	`transferFrom` can be replaced by a `transfer`

Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

In the `withdrawERC20` function, called when the contract's owner intends to withdraw the remaining deposited tokens, the `transferFrom` function is utilized to transfer the tokens from the contract to the recipient. However, this operation could be simplified by replacing `transferFrom` with a straightforward `transfer` function call.

Alleviation

The team fixed the issue at commit hash [Eac3c28c6ce68df38080bf70a0e1850dd44c4575](Eac3c28c6ce68df38080bf70a0e1850dd44c4575).

INFO-2	`multiplier` could be immutable
Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

The `calculateAmount()` function converts an amount from the 8 decimal accuracy used by BTC to the decimals of the tokens by multiplying it by the multiplier.

```
    uint256 multiplier = 10 ** (token.decimals() - 8);
```

However, the number of decimals, and consequently the `multiplier`, are fixed (at least for standard ERC20 tokens). Therefore, there is no need to compute it repeatedly on each call to `calculateAmount`.

Recommendation

We suggest declaring the `multiplier` as immutable and computing it once in the constructor. This approach will save gas by avoiding redundant calculations.

Alleviation

The team fixed the issue at commit hash [0eb35d22a91ac7b0f6d9d1769c6bc2c2c0c2528b](0eb35d22a91ac7b0f6d9d1769c6bc2c2c0c2528b).

INFO-3	Missing sanity check that `_txProofDifficultyFactor` is non zero
Contract(s)	`OnrampFactory.sol`
Status	**Resolved**

Description

To prevent incorrect configurations by mistake, we suggest adding a check in the constructor to ensure that the provided value for `_txProofDifficultyFactor` is non-zero

Alleviation

The team fixed the issue at commit hash [7c7b4bef36796846ad2c657083f368d6b25e6c1c](7c7b4bef36796846ad2c657083f368d6b25e6c1c).

INFO-4	Allow `_dustThreshold` take the minimum value
Contract(s)	`Onramp.sol`
Status	**Resolved**

Description

In the `setDustThreshold` function, callable only by the contract's owner, the `_dustThreshold` variable can be set to any value strictly greater than the minimum `_DEFAULT_DUST_THRESHOLD`, which is set during deployment. We suggest replacing > with >= to allow users to reset this variable to its original value if they wish to do so.

Alleviation

The team fixed the issue at commit hash [f1bdd58ca43e6902336370a6b7cc5f2115905388](f1bdd58ca43e6902336370a6b7cc5f2115905388).

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to

help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.