BitcornOFT Audit Report

Version 1.0

Windhustler

Corn Blockchain OFT contracts Audit Report

Windhustler

December 20, 2024

Introduction

A time-boxed security review of the Corn blockchain OFT contracts was done by **Windhustler**, focusing on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About Windhustler

Windhustler is an independent smart contract security researcher. Having extensive experience in developing and managing DeFi projects holding millions in TVL, he is putting his best efforts into security research & reviews. Check his previous work here or reach out on X @windhustler.

About Corn blockchain

Corn is a novel blockchain network built on the Arbitrum Orbit stack, designed to harness Bitcoin's value with Ethereum's computational power. With its hybrid tokenized Bitcoin (BTCN) as its gas token, Corn offers a unique, secure, and sustainable way to maximize Bitcoin's potential.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Security Assessment Summary

***review commit hash* - 250a6f164ffbaa2f1b441d4f0c3370e05c53280f**

Scope

The following smart contracts were in the scope of the audit:

- SwapOAppComposer.sol
- StandardOFTAdapterUpgradeable.sol
- UsdcAdapterUpgradeable.sol

Findings Summary

ID	Title	Severity	Status
L-01	Buffer is insufficient to execute the catch block	Low	Fixed
L-02	`msg.value` paid with `lzCompose` is lost	Low	Acknowledged
L-03	Wrongly encoded data for the `lzCompose` function results in loss of funds for smart contract wallets	Low	Acknowledged
L-04	Bridging USDC to `address(0)` will result in loss of funds	Low	Fixed
L-05	Bridging USDC to a blacklisted address will result in loss of funds	Low	Acknowledged

Detailed Findings

[L-01] Buffer is insufficient to execute the catch block

Context

- SafeCallMinGas.sol#L12-L14

Description

The `hasMinGas` function is used to ensure that the external call has sufficient gas to execute while leaving a buffer to execute the catch block in case of an error, as the external call to swap facility is wrapped in a **`try`**/**`catch`** block.

```
## SwapOAppComposer.sol

  require(hasMinGas(SWAP_OUT_MAX_GAS_BUDGET, 0), "Must have sufficient gas for swap");

  // Step 3: Execute swap with gas limit
  debtToken.approve(address(swapFacility), _debtIn);
  try swapFacility.swapExactDebtForCollateral{gas: SWAP_OUT_MAX_GAS_BUDGET}(
      _debtIn, _minCollateralOut, _receiver, block.timestamp
  ) {} catch {
      // Swap failed - need to revoke approval since it won't be automatically reverted
      debtToken.approve(address(swapFacility), 0);
      debtToken.safeTransfer(_receiver, _debtIn);
  }
```

The assumption here is that if `hasMinGas` returns true the `lzCompose` function will execute successfully. The issue is that the reserved buffer amount of gas is only `10_000` gas so if the try block consumes all the `SWAP_OUT_MAX_GAS_BUDGET` gas, the catch block will not be executed.

Also, the comment in the `hasMinGas` function is incorrect as it states that the buffer is `40_000` gas, but it's actually `10_000` gas.

Recommendation

Increase the `ADDITIONAL_CALL_BUFFER` to 50_000 gas.

Project

Mitigated in e5e90b19f97a4531357132b73ec5bfcd45844b46 as per recommendation by increasing ADDITIONAL_CALL_Buffer in SafeMinGas to 60k from 10k to ensure the user has enough gas to execute the approve + transfer if the swap facility swap fails. Adding a little extra buffer for unexpected future changes to BitcornOFT. Should be kept in mind for future upgrades.

Resolution

Fixed in e5e90b19f97a4531357132b73ec5bfcd45844b46.

[L-02] `msg.value` paid with `lzCompose` is lost

Context

- SwapOAppComposer.sol#L51-L57

Description

User's can pay for additional `msg.value` to be sent with the `lzCompose` function call. It's a problem as this value will just be left in the `SwapOAppComposer` contract without the ability to be retrieved by the user.

Recommendation

Consider a design where this value is refunded to the user in case of an error.

Project

Acknowledged, user error is fundamentally possible in LZ compose. The SwapFacilityCrosschainRedemptionZap is introduced to make parameter construction simpler and safer.

Resolution

The issue is acknowledged and handled by introducing a periphery contract to encode parameters.

[L-03] Wrongly encoded data for the `lzCompose` function results in loss of funds for smart contract wallets

Context

- SwapOAppComposer.sol#L71-L83

Description

In case when the sender of the OFT is a smart contract wallet, he might not control the same address on the destination chain. If the sender doesn't encode the data properly the tokens get transferred to the sender's address from the source chain.

```
## SwapOAppComposer.sol

    function lzCompose(
        address _from,
        bytes32, /*_guid*/
        bytes calldata _message,
        address, /*Executor*/
        bytes calldata /*Executor Data*/
    ) external payable override {
        require(_from == oApp, "!oApp");
        require(msg.sender == endpoint, "!endpoint");
        // Extract the composed message from the delivered message using the MsgCodec

        uint256 debtIn = OFTComposeMsgCodec.amountLD(_message);

        // Could be desynched when dealing with SC wallet, so we ask the sender
        // TODO: Sanitize more on the send side for the token?
        address recipient;
        uint256 minCollateralOut;
        try this.decodeInput(OFTComposeMsgCodec.composeMsg(_message)) returns (address _recipient, uint256 _minCollateralOut) {
            recipient = _recipient;
            minCollateralOut = _minCollateralOut;
        } catch {
            // Failure case, we don't have sufficient data
            // Send tokens to original sender
            // NOTE: This causes losses to SC Wallet
            // SC Wallets MUST encode the data properly before using this contract

            // Grab compose sender
>>>            address receiver = OFTComposeMsgCodec.bytes32ToAddress(OFTComposeMsgCodec.composeFrom(_message));

            // Send to them
            debtToken.safeTransfer(receiver, debtIn);

            return;
        }
```

This will result in loss of funds for smart contract wallets.

Recommendation

Consider enforcing the structure of the `composeMsg` on the sending chain, i.e. enforcing the sender to specify the receiver address in case of failure inside the `lzCompose`.

Project

Acknowledged, user error is fundamentally possible in LZ compose. Recommendation is used in new SwapFacilityCrosschainRedemptionZap to make parameter construction simpler and safer.

Resolution

The issue is acknowledged and handled by introducing a periphery contract to encode parameters.

[L-04] Bridging USDC to `address(0)` will result in loss of funds

Context

- UsdcAdapterUpgradeable.sol#L73

Description

If a user sends tokens from Ethereum to the Corn blockchain and sets `SendParam.to` to `address(0)`, and then calls the `OFTCoreUpgradeable::send` function with this parameter, the function will not revert. On the Corn blockchain, the`UsdcAdapterUpgradeable::_credit` function will be invoked with `_to` set to `address(0)`. When the function attempts to execute `IFiatToken(address(innerToken)).mint(_to, _amountLD)`, it will fail because minting to the zero address is not allowed. This causes the tokens to be locked in the adapter contract on the Ethereum blockchain.

Recommendation

If the `to` address is set to `address(0)`, mint the tokens to the `address(0xdead)` address.

Project

Mitigated in d254583564a4eab22b5039818f0bc94cfc38134d as per recommendation if the to address is set to address(0), mint the tokens to the address(0xdead) address.

Resolution

Fixed in d254583564a4eab22b5039818f0bc94cfc38134d.

[L-05] Bridging USDC to a blacklisted address will result in loss of funds

Context

- UsdcAdapterUpgradeable.sol#L20
- StandardOFTAdapterUpgradeable.sol#L17

Description

USDC has a blacklist of addresses that you can't mint or transfer to. If a user sends tokens from Ethereum to the Corn blockchain or vice versa and sets `SendParam.to` to a blacklisted address, the function will not revert on the sending side but will be non-executable on the receiving side. Both scenarios will result in excess funds being locked in the adapter contract on Ethereum.

Recommendation

Consider transferring the tokens to the `_to` address within a try/catch block. If the transfer fails we can assume that the address is blacklisted and the amount can be stored in an internal mapping to keep track of all the funds that were lost. Although less likely to happen on L2, consider wrapping the `mint` call in a 'try/catch and keep account of all the failed minting attempt amounts.

Project

Acknowledged. Working under the assumption that the main reason to track failed mints onchain is for offchain accounting purposes, these failed mints can be tracked via events and tx and left as an exercise to the reader. Noting if a user becomes unblacklisted, the transaction will no longer be redeemable in the recommendation, whereas I believe it would be finishable via permissionless execution in the current paradigm.

Resolution

The issue is acknowledged.

Informational issues

- decodeInput function can be declared as pure.
- `srcEid` can be required in the `lzCompose` function to always match Corn eid to add an extra layer of safety. This can be fetched from the `composeMsg`.
- StandardOFTAdapterUpgradeable.sol and, UsdcAdapterUpgradeable.sol contracts are upgradeable and prior to migration to native USDC should be upgraded with appropriate logic to accommodate the specification.