code4rena

Corn

Smart Contract Security Assessment

Version 1.0

Audit dates: Nov 27 — Nov 29, 2024

Audited by: EV_om
3DOC

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Corn

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. Secured by both Bitcoin (via Babylon) and Ethereum staking, Corn ensures long-term alignment through its innovative popCORN incentives model.

2.2 Scope

Repository	usecorn/bitcorn-oft/
Commit Hash	df6dd98acea31bf3f390ad92d5f489bb21068247

2.3 Audit Timeline

DATE	EVENT
Nov 27, 2024	Audit start
Nov 29, 2024	Audit end
Dec 12, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	4
Low Risk	5
Informational	0
Total Issues	9

3. Findings Summary

ID	DESCRIPTION	STATUS
M-1	SwapFacility does not handle collaterals that charge fees on transfer	Acknowledged
M-2	SwapFacility does not handle collaterals that do not revert on failures	Resolved

M-3	WrappedBitcornNativeOFTAdapter is pausable but can't be paused	Resolved
M-4	Admin setters in SwapFacility revert when paused	Resolved
L-1	Unnecessary user-provided parameters could lead to lost funds	Acknowledged
L-2	Reentrancy window in `swapExactCollateralForDebtAndLZSend`	Resolved
L-3	`sendParam.amountLD` being overwritten is unnecessary and confusing	Acknowledged
L-4	Superfluous `whenNotPaused` modifiers	Acknowledged
L-5	WrappedBitcornNativeOFTAdapter.send calls can be DoS'ed via wrapped token donations	Acknowledged

4. Findings

4.1 Medium Risk

A total of 4 medium risk findings were identified.

[M-1] SwapFacility does not handle collaterals that charge fees on transfer

Severity: Medium	Status: Acknowledged

Context:

- SwapFacility.sol

Description: Within the audit scope, we are considering the case of ERC20 contracts that charge fees on transfer.

While the Ethereum and Corn Bitcorn ERC20 contracts are within the scope, the collaterals aren't and if we consider the case of collaterals having this "weird" behavior of charging fees on transfer, we can see how `SwapFacility` could be vulnerable because it does not check for the amount effectively transferred:

```
File: SwapFacility.sol
169:     function swapExactCollateralForDebt(uint256 collateralIn,
uint256 debtOutMin, address to, uint256 deadline)
---
175:     {
---
183:         debtToken.transfer(to, debtOut);
184:         collateralToken.transferFrom(msg.sender, address(vault),
collateralIn);
185:     }
186:
---
198:     function swapExactCollateralForDebtAndLZSend(
---
204:     )
---
212:     {
---
224:         collateralToken.transferFrom(msg.sender, address(vault),
collateralIn);
225:     }
```

The consequence of this is that if a collateral has fees enabled on transfer, these fees will impact the `swapIn` and `swapOut` fees, and if these are low enough, the collateralization of the debt token.

Recommendation: In "swapExactCollateralForDebt" operations, we recommended first transferring tokens to the vault, checking its balance delta, and calculating output debt tokens according to the amount of tokens the vault actually received instead of those sent.

Corn: Acknowledged, will include fee on transfer incompatibility in usage requirements for this version and will keep this accounting style in future iterations.

[M-2] SwapFacility does not handle collaterals that do not revert on failures

Severity: Medium	Status: Resolved

Context:

- SwapFacility.sol

Description: Within the audit scope, we are considering the case of ERC20 contracts that don't revert on failures (i.e. return `false`).

While the Ethereum and Corn Bitcorn ERC20 contracts are within the scope, the collaterals aren't and if we consider the case of collaterals having this "weird" behavior of not reverting on failures, we can see how `SwapFacility` could be vulnerable because it doesn't check for return values to be `true`:

```
File: SwapFacility.sol
169:     function swapExactCollateralForDebt(uint256 collateralIn,
uint256 debtOutMin, address to, uint256 deadline)
---
175:     {
---
183:         debtToken.transfer(to, debtOut);
184:         collateralToken.transferFrom(msg.sender, address(vault),
collateralIn);
185:     }
186:
---
198:     function swapExactCollateralForDebtAndLZSend(
---
204:     )
---
212:     {
---
224:         collateralToken.transferFrom(msg.sender, address(vault),
collateralIn);
225:     }
---
232:     function swapDebtForExactCollateral(uint256 collateralOut,
uint256 debtInMax, address to, uint256 deadline)
---
238:     {
---
246:         debtToken.transferFrom(msg.sender, address(this), debtIn);
```

```
247:         collateralToken.transferFrom(address(vault), to,
collateralOut);
248:     }
```

Recommendation: We recommend using the OpenZeppelin's `SafeERC20` library for protecting `transferFrom` operations from failed calls that return `false` instead of reverting.

Corn: Agree, mitigated in commit @d1a8ead457...

Zenith: Verified

[M-3] WrappedBitcornNativeOFTAdapter is pausable but can't be paused

Severity: Medium	Status: Resolved

Context:

- WrappedBitcornNativeOFTAdapter.sol

Description: The `WrappedBitcornNativeOFTAdapter` is a `Pausable` contract, implementing a `whenNotPaused` check to the `_update` method, with the intention of preventing on-chain and cross-chain transfers when paused.

The `WrappedBitcornNativeOFTAdapter` contract gets plausibility from its parent `ERC20PausableUpgradeable` contract; quoting the natspec of this contract:

```
File: ERC20PausableUpgradeable.sol
17:  * IMPORTANT: This contract does not include public pause and unpause
functions. In
18:  * addition to inheriting this contract, you must define both
functions, invoking the
19:  * {Pausable-_pause} and {Pausable-_unpause} internal functions, with
appropriate
```

its children are expected to implement `pause` and `unpause` functions like done in `BitcornOFT`. `WrappedBitcornNativeOFTAdapter` lacks these methods.

Recommendation: We recommend adding to `WrappedBitcornNativeOFTAdapter` similar methods as `BitcornOFT`:

```
File: BitcornOFT.sol
81:     /// @notice Pauses all token transfers.
82:     function pause() public requiresAuth {
83:         _pause();
84:     }
85:     /// @notice Unpauses all token transfers.
86:     function unpause() public requiresAuth {
87:         _unpause();
88:     }
```

Corn: Agree that pausing should be included, added in @40ad93b70ed6...

Zenith: Verified

[M-4] Admin setters in SwapFacility revert when paused

Severity: Medium	Status: Resolved

Context:

- SwapFacility.sol

Description: `SwapFacility` is a `Pausable` contract, which has a centralized governance managed by `Authority`.

The following admin setters of `SwapFacility` are marked both `whenNotPaused` and `requiresAuth`:

```
File: SwapFacility.sol
378:     function setSwapInFeeRate(uint256 _feeRate) public requiresAuth
whenNotPaused {
---
386:     function setSwapOutFeeRate(uint256 _feeRate) public requiresAuth
whenNotPaused {
---
404:     function setSwapInEnabled(bool _swapInEnabled) external
requiresAuth whenNotPaused {
---
411:     function setSwapOutEnabled(bool _swapOutEnabled) external
requiresAuth whenNotPaused {
---
418:     function setCrosschainSwapInEnabled(bool
_crosschainSwapInEnabled) external requiresAuth whenNotPaused {
---
425:     function setMintCap(uint256 _mintCap) external requiresAuth
whenNotPaused {
---
434:     function setPreMintedDebtTarget(uint256 _preMintedDebtTarget)
external requiresAuth whenNotPaused {
---
441:     function setFeeRecipient(address _feeRecipient) external
requiresAuth whenNotPaused {
```

This can be problematic because in case something bad happens, the typical operations admins do are:

- immediately pause the contract
- investigate the issue

- possibly, plan for a limited functionality un-pause
- modify the contract configuration or upgrade it to fix the issue
- unpause the contract to resume operations with new parameters

However, in the current implementation, it is not possible for an admin to make any calls while the contract is paused.

Recommendation: We recommend removing `whenNotPaused` from the `requiresAuth` setters above.

Corn: Agree, mitigated in @e11d00610663e1...

Zenith: Verified

4.2 Low Risk

A total of 5 low risk findings were identified.

[L-1] Unnecessary user-provided parameters could lead to lost funds

Severity: Low	Status: Acknowledged

Context:

- SwapFacility.sol
- WrappedBitcornNativeOFTAdapter.sol

Description: Both `SwapFacility.swapExactCollateralForDebtAndLZSend()` and `WrappedBitcornNativeOFTAdapter.send()` take a `SendParam` struct as parameter, which contains a `dstEid` field denoting the destination Endpoint ID. This variable can however in both cases only take one possible value - either Corn or Ethereum - and effectively only allows users to lose their funds.

Recommendation: If user-friendliness of the contracts is a concern, and particularly if it is expected for users to make custom calls (not via a protocol-controlled frontend), it is recommended to remove all such "dangling" parameters and to perform validation on all other user-provided values to ensure they fall within their expected ranges.

Corn: Acknowledged, maintaining existing interface for more flexibility of use in future scenarios at expense of param construction convenience and safety.

[L-2] Reentrancy window in `swapExactCollateralForDebtAndLZSend`

Severity: Low	Status: Resolved

Context:

- SwapFacility.sol

Description: In `swapExactCollateralForDebtAndLZSend()`, there is a reentrancy window where the `refundAddress` is called before the collateral transfer is complete, leading to temporarily inconsistent accounting state. While this doesn't appear to have serious consequences (only `collectFees()` behaves differently if called within the inconsistent state, and it only leads to fee accounting being deflated), it represents an unnecessary reentrancy risk.

Recommendation: Reorder the operations to transfer the collateral before calling the external IOFT send function to eliminate any possibility of reentrancy during an inconsistent state.

Corn: Mitigated by moving swap & send functionality to external zap contract

Zenith: Verified

[L-3] `sendParam.amountLD` being overwritten is unnecessary and confusing

Severity: Low	Status: Acknowledged

Context:

- SwapFacility.sol

Description: The `swapExactCollateralForDebtAndLZSend` function accepts a `SendParam` struct as input but then overwrites its `amountLD` field. This is poor API design that could lead to confusion and potential errors, as callers may unnecessarily populate a field that will be ignored.

Recommendation: Instead of accepting and modifying the full `SendParam` struct, the function should accept the individual required fields (`dstEid`, `to`, `minAmountLD`) as parameters and construct the `SendParam` struct internally with the calculated `amountLD`. Ideally, rename the parameters to reflect their purpose within the call.

This provides a cleaner and more intuitive API.

Corn: Acknowledged, agree with the issue but maintaining original design to keep external construction of LZ parameters. However, this logic was changed to an external zap

[L-4] Superfluous `whenNotPaused` modifiers

Severity: Low	Status: Acknowledged

Context:

- BitcornOFT.sol

Description: There are multiple instances of `whenNotPaused` usage that are unnecessary because the same modifier is reached further down the call flow. Examples of this are `mint()`, `mintTo()` and `burn()` in BitcornOFT, which reach `whenNotPaused` in `_beforeTokenTransfer` -> `ERC20PausableUpgradeable._beforeTokenTransfer()`

Recommendation: Remove the duplicate modifiers for codebase coherence and slightly lower gas costs.

Corn: Acknowledged, agree but maintaining for external clarity.

[L-5] WrappedBitcornNativeOFTAdapter.send calls can be DoS'ed via wrapped token donations

Severity: Low	Status: Acknowledged

Context:

- WrappedBitcornNativeOFTAdapter.sol

Description: The `WrappedBitcornNativeOFTAdapter.send` function contains two branches depending on the sender's balance in wrapped tokens:

```
File: WrappedBitcornNativeOFTAdapter.sol
152:     function send(SendParam calldata _sendParam, MessagingFee
calldata _fee, address _refundAddress)
153:         external
154:         payable
155:         override
156:         returns (MessagingReceipt memory msgReceipt, OFTReceipt
memory oftReceipt)
157:     {
158:         uint256 msgSenderBalance = balanceOf(msg.sender);
159:         MessagingFee memory feeWithExtraAmount =
MessagingFee({nativeFee: _fee.nativeFee, lzTokenFee: _fee.lzTokenFee});
160:
161:         if (msgSenderBalance < _sendParam.amountLD) {
162:             if (msgSenderBalance + msg.value < _sendParam.amountLD)
{
163:                 revert InsufficientMessageValue();
164:             }
165:
166:             // user can cover difference with additional msg.value
ie. wrapping
167:             uint256 mintAmount = _sendParam.amountLD -
msgSenderBalance;
168:             _mint(address(msg.sender), mintAmount);
169:
170:             // update the messageFee to take out mintAmount
171:             feeWithExtraAmount.nativeFee = msg.value - mintAmount;
172:         } else if (msg.value != feeWithExtraAmount.nativeFee) {
173:             revert NotEnoughNative(msg.value);
174:         }
```

The `if` branch handles the case of the sender having insufficient balance: it ignores the input `_fee.nativeFee` and it overwrites it at L171;

on the contrary, in the `else` branch, the input `_fee.nativeFee` is required to be exactly `msg.value`.

We imagine the situation of a user calculating the inputs to make a call that will pass through the `if` branch: they will use some of their balance, and integrate it with some extra `msg.value` that depend on their balance, to make ends meet with the desired `nativeFee`.

Now a malicious actor can frontrun this call and donate tokens to the caller; this will force execution into the `else` branch, and because the check at L172 requires exact equality, the call is likely to revert.

Recommendation: We recommend improving the UX by always receiving explicit instruction from the user:

- always use the `_fee.nativeFee` specified by the user, regardless of `msgSenderBalance` and `msg.value`
- compute `extra = msg.value - nativeFee - amountLD`
- burn balance if `extra` is negative (this burn operation fails if balance is insufficient, as expected)
- if `extra` is positive, it can be either refunded to the caller or minted as wrapped to their account

Corn: Acknowledged, agree with the change but will retain existing codebase in this release for code maturity reasons.