Bitcorn OFT

Security Review

Cantina Managed review by:
D-Nice, Lead Security Researcher
Sujith Somraaj, Security Researcher

December 14, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Corn is a layer 2 network focused on revolutionizing the capital efficiency of Bitcoin as a nascent asset class. Designed to provide scalable infrastructure that leverages Ethereum in a manner that allows for the secure management of billions of dollars in liquidity with low transactional costs secured by the Bitcoin L1.

From Nov 25th to Nov 29th the Cantina team conducted a review of bitcorn-oft on commit hash 6e0e319c. The team identified a total of **25** issues in the following risk categories:

- Critical Risk: 1
- High Risk: 1
- Medium Risk: 4
- Low Risk: 7
- Gas Optimizations: 1
- Informational: 11

The scope of the review included all the smart contracts present in the repository at the reviewed commit hash.

3 Findings

3.1 Critical Risk

3.1.1 Manipulated `sendParam` in `swapExactCollateralForDebtAndLZSend` function allows unauthorized token minting

Severity: Critical Risk

Context: SwapFacility.sol#L197

Description: The `swapExactCollateralForDebtAndLZSend` function allows malicious users to mint arbitrary amounts of debt tokens by manipulating LayerZero send parameters, bypassing collateral requirements.

This issue exists because the function fails to validate that the cross-chain send amount (`sendParam.amountLD`) matches the calculated debt output (`debtOut`).

An attacker can:

- Provide valid `collateralIn` amount.
- Set `sendParam.amountLD` to any arbitrary value (maximum pre-minted debt cap).
- Execute cross-chain transfer with an inflated amount.
- Receive excess minted tokens on the destination chain.

Proof of Concept: The following proof of concept demonstrates the issue:

- ETH mainnet fork (`chainId`: 1).
- Arbitrum fork (`chainId`: 42161).
- Deploys `BitcornOFT` on ETH and `WrappedBitcornNativeOFTAdapter` on Arbitrum:

```
SendParam memory sendParam = SendParam(
    30110,  // destination chain
    bytes32(uint256(uint160(deployer))),  // recipient
    collateralIn,  // amount to send - can be manipulated
    0,  // minAmount
    OptionsBuilder.encodeLegacyOptionsType1(200_000),
    bytes(""),
    bytes("")
);

swapFacility.swapExactCollateralForDebtAndLZSend{value: 1 ether}(
    1e18,  // Only deposits 1 token as collateral
    sendParam,  // But requests much larger amount in sendParam
    lzFee,
    deployer,
    block.timestamp
);
```

`sendParam.amountLD` is not validated against the actual collateral-backed debt amount (debtOut), allowing arbitrary amounts to be minted on the destination chain.

Recommendation: Consider fixing the issue by following one of the following recommendations:

- Construct the `sendParam` inside the function rather than accept arbitrary inputs from the user; or...
- Validate the `sendParam` against the `debtOut` values to ensure the accuracy of the cross-chain message.

Bitcorn: Fixed in f53bd74f.

Cantina Managed: Verified fix. The `sendParam.amountLD` is now set to `debtOut` instead of the user-specified value, preventing this issue.

3.2 High Risk

3.2.1 Highly permissioned mechanisms in architecture, could allow arbitrary minting and collateral siphon by malicious owner/gov

Severity: High Risk

Context: BitcornOFT.sol#L52-L70, Auth.sol#L24-L46, SwapFacility.sol#L226-L248

Description: The current base architecture with authorities that can grant finegrained roles and associated upgradeable contracts, makes the vaults have to be considered as custodial on-chain entities, in which the owner must be trusted to not act maliciously or against the interests of collateral providers.

The owner and authorities of the connected `BitcornOFT` `Auth` can grant any address, either contract or EOA, privileges ranging from further authority grants to minting. Potentially allowing for the ability to mint the necessary amount of `debtTokens` needed to drain all the vaults of their collateral. This would collapse `Bitcorn` value as a sacrifice for having gained all the collateral value from vaults. Other potential attacks exist too where AMMs or the like have `Bitcorn` dumped and then pull collateral.

These vectors exists as long as there is:

- A non-null `owner` in the `Authority` contract set in `BitcornOFT`.
- Any addresses authorized to grant `setAuthority` or `mint` (or variants of it) in `Authority`.
- Mutable contracts part of the architecture.

A number of these aspects are in play to allow the project to do updates which may include fixes or upgrades, and expansion of the protocols, but do put users at risk. In the hands of a benevolent owner, these tools can help save and recover user funds in case of issues, and the series of contracts are dependent on multiple moving parts for their operation.

Documentation has noted that Governor contracts are intended to be used with some safeguards, however, these were not within scope of the audit, and a Governor contract would still have this ability, unless some hardcoded minter/upgrade limits were in place, albeit with some execution delay lowering the overall impact and chance of success.

Recommendation: The ideal solution to mitigate this concern of centralized authority and trust which could be trivially abused is on a finalized and established version of `Bitcorn` to discard any `Authority` granters, and ensure `mint/burn` and similar are only permitted to immutable `SwapFacility` contracts. All involved contracts should be immutable, and could be considered so if the authority to upgrade is discarded.

Another solution that could retain some of the flexibilities of auth granting and upgradeable contracts is to utilize only a or a series of contracts with a commit & execute schema for auth & updates, with a significant timelock on the execution in place (at least several days to allow for it to be noticed and provide sufficient LayerZero cross-chain transfer time), which would at least allow some vigilant users to notice exploit and provide recovery for themselves and hopefully others before an exploit by malicious owner could be executed.

Bitcorn: Acknowledging fundamental centralization risks, will focus on increasing decentralization, transparency, and otherwise mitigating admin risk and open to ideas.

Cantina Managed: Acknowledged.

3.3 Medium Risk

3.3.1 Uncollected fees may be irrevocably burnt when calling `SwapFacility.setMintCap()` to be below fee

Severity: Medium Risk

Context: SwapFacility.sol#L352-L359, SwapFacility.sol#L364-L370, SwapFacility.sol#L425-L428

Description: Setting `debtMintCap` to 0 or any value that is below the fee amount via `setMintCap()`, then allows anyone calling `burnExcessDebt()` to burn all uncollected fees for the 0 case or a portion depending on the uncollected fee differential and new cap. The burnt fees would be irrecoverable with the contracts with their current logic, and the associated collateral stuck in the vault.

There may be potential recovery options via upgrades of the contracts and hardcoding a recovery for the lost fees.

Recommendation: By ensuring a specific order of functions are called, this scenario can be avoided. `collectFees()` must be called before any `debtMintCap` alterations, ideally via a multicall in a single transaction, or for the contract to be in a paused state during this sequence of calls. If going the paused route, the `whenNotPaused` modifer should be omitted, so fees can be collected in the paused state, ensuring no intermediate swaps happen that may yield fees in between, which would be lost in a 0 set scenario.

Bitcorn: Mitigated in new fee design where fees are explicitly managed in storage variable, see commit ab24673b.

Cantina Managed: Verified fix.

3.3.2 `SwapFacility.swapExactCollateralForDebt()` calls by unvigilant users may be owner frontrunnable for value capture via fees

Severity: Medium Risk

Context: SwapFacility.sol#L161-L294, SwapFacility.sol#L376-L390

Description: A malicious owner that has access to the setters `setSwapInFeeRate()` or `setSwapOutFeeRate()`, may frontrun calls to `SwapFacility.swapExactCollateralForDebt()` or `SwapFacility.swapExactCollateralForDebtAndLzSend()` or `swapDebtForExactCollateral()` done by unvigilant users without appropriate `debtOutMin` or `debtOutMax` set, allowing for up to 100% value capture of the collateral in the most ideal scenario, which would require a setting of 0 on `debtOutMin`, and the owner to frontrun the user by changing `setSwapInFeeRate()` to effectively 100% or `MAX_BPS`.

Since most users would likely utilize an owner provided UI for contract interaction, the UI could be trivially set to send such parameters, and the biggest challenge would be to achieve the frontrun.

Recommendation: Protections are already technically in place via the noted min max parameters, but users will need to be vigilant and ensure these are set to safe expected thresholds prior to sending significant collateral, within the transaction being signed.

Additional safeguards would be implementing timelocks or delays on access of these permissioned functions by the owner (ideally within the contract itself, as timelocked multisigs could also easily transfer authority to entities without such restrictions), thereby reducing the chances of frontrun, in most cases close to 0. Furthermore, there is unlikely a case where a fee of 100% is necessary, and the setters should be limited to a lower sane fee threshold. Having these implemented, would also protect unvigilant users to a degree.

Bitcorn: Partially mitigated by setting max fee to 10% in commit 7c66b722. Will be additionally mitigated by relevant admin permissions going behind timelock.

Cantina Managed: Verified fix.

3.3.3 Inconsistent native fee validation leading to transaction failures

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The `WrappedBitcornNativeOFTAdapter.sol` contract contains contradictory native fee validation logic between its `send` and `_payNative` functions, causing unnecessary transaction failures.

```
// In _payNative:
if (msg.value < _nativeFee) revert NotEnoughNative(msg.value);

// In send:
if (msg.value != feeWithExtraAmount.nativeFee) {
    revert NotEnoughNative(msg.value);
}
```

Since the refund address is explicitly forwarded to the LayerZero endpoint, any excessive gas fees will be refunded automatically.

Proof of Concept: The following proof of concept demonstrates transaction reverts, though the user overpays:

```
function test_e2e() external {
   vm.startPrank(deployer);
   vm.selectFork(forkId[ARBI]);
   deal(deployer, 100 ether);

   wrapper.deposit{value: 1 ether}();

   SendParam memory sendParam = SendParam(30101, bytes32(uint256(uint160(deployer))), 1 ether, 0,
↪  OptionsBuilder.encodeLegacyOptionsType1(200_000), bytes(""), bytes(""));
   MessagingFee memory lzFee = MessagingFee(1 ether, 0);
   wrapper.send{value: 2 ether}(sendParam, lzFee, address(deployer));
}
```

Impact:

- Valid transactions revert unnecessarily in certain paths.
- Users overpaying fees have transactions fail in certain paths.

Recommendation: The gas fee validation is done inside the `_lzSend` function; hence, remove this redundant validation to ensure consistency and avoid unexpected behavior. If you still think, reverting earlier in the stack is relevant, then modify the `send` function to use consistent validation:

```
// In send:
if (msg.value < feeWithExtraAmount.nativeFee) {
    revert NotEnoughNative(msg.value);
}
```

Bitcorn: Fixed in commit 69d1ec6c.

Cantina Managed: Verified fix.

3.3.4 Inaccessible pause mechanism in `WrappedBitcornNativeOFTAdapter`

Severity: Medium Risk

Context: WrappedBitcornNativeOFTAdapter.sol#L27

Description: The `WrappedBitcornNativeOFTAdapter.sol` contract inherits `ERC20PausableUpgradeable` but fails to implement the `pause()` and `unpause()` functions.

Due to the absence of these functions, pausing the protocol during emergencies is not possible. Additionally, the `whenNotPaused` modifier, added to the `impossible` function to restrict transfers after a pause, will become ineffective and unnecessary.

```
contract WrappedBitcornNativeOFTAdapter is
    ERC20PausableUpgradeable,
    // ...other inheritances
```

Recommendation: Consider adding the `pause()` and `unpause()` functions to enable the pause functionality:

```
function pause() external requiresAuth {
    _pause();
}

function unpause() external requiresAuth {
    _unpause();
}
```

Bitcorn: Fixed in 40ad93b7

Cantina Managed: Verified fix.

3.4 Low Risk

3.4.1 Avoid tightly packed variables as initial declarations in upgradeable contract

Severity: Low Risk

Context: SwapFacility.sol#L42-L49, SwapFacility.sol#L64-L66

Description: If declaring smaller sized types that get tightly packed as the initial declarations in a contract, they may end up packing unexpectedly with ungapped dependency contracts that end with smaller types. This can lead to storage slot clashes and conflicts upon attempting upgrades.

Recommendation: Introduce a pre-gap, or ensure a full-sized type is initially declared. Also consider the potential of utilizing a namespaced storage layout, which would give finegrained control over the packing.

For minimal changes, simply move the codeblock from L45-L49 to L65, so that the `feeRecipient` declaration follows it, and thereby all those variables will be packed into one storage slot for additional potential gas savings on storage loads, and avoid potential unexpected packing with dependencies.

Bitcorn: Mitigated by moving tightly-packed variables after full slot variables in commit 518ac9db.

Cantina Managed: This issue itself is fixed, although I recommend the more optimized fix; i.e. you would ideally pack them with `feeRecipient`, like has been done in the `simpleSwapFacility`, that way they all take up one shared storage slot, and then you can implement a multi-setter function for all those variables which will cost one SSTORE opcode only for setting potentially all.

3.4.2 Utilize `SafeERC20.safeTransferFrom()` when interacting with 3rd-party tokens

Severity: Low Risk

Context: SwapFacility.sol#L182, SwapFacility.sol#L218, SwapFacility.sol#L245

Description: It is best practice to utilize the "safe" variant of functions when interacting with unknown/untrusted/3rd Party ERC20 token implementations. Namely with regards to the `safeTransferFrom()` variant which avoid potential issues or irregular behaviours that may arise from varying ERC20 implementations, such as missing return value bugs, leading to lost or inaccessible tokens.

Recommendation: Utilize `safeTransferFrom()` whenever dealing with tokens not directly within the contract architecture, and consider other "safe" function variants whenever possible.

Bitcorn: `safeTransferFrom` used in interaction with collateralToken as described, mitigated in commit d1a8ead4.

Cantina Managed: Verified fix.

3.4.3 Incorrect LayerZero token fee handling

Severity: Low Risk

Context: WrappedBitcornNativeOFTAdapter.sol#L154

Description: The `swapExactCollateralForDebtAndLZSend` function in `SwapFacility.sol` and `send` function in `WrappedBitcornNativeOFTAdapter.sol` accepts a `MessagingFee` parameter with `lzTokenFee` but does not implement any logic to handle token fees.

In the LayerZero protocol, users can cover the costs of cross-chain messaging using either the native token of their blockchain (for example, ETH on Ethereum) or the LayerZero token. The `lzTokenFee` value is used when users choose to pay for cross-chain messaging with the LayerZero token. However, to enable this option, the tokens must be approved and transferred from the user to the `Endpoint.sol` contract.

Recommendation: Consider either of the following recommendations:

- Remove token fee support: Replace `MessagingFee` with simple uint256 nativeFee and construct the `MessagingFee` internally.
- Implement token fee handling: Add proper token transfers and approvals for LayerZero fees.
- Or add validations to ensure the value is always zero:

```
if(msg.value < lzFee.nativeFee || lzFee.lzTokenFee != 0) revert InvalidFee();
```

Bitcorn: Fixed in commit d217ea3b

Cantina Managed: Verified fix.

3.4.4 `SwapFacility` token buffer may be exhausted if `collectFees()` is frontrun by sufficient donation, leading to remediable DoS

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `SwapFacility.collectFees()` function can be piggybacked as a mechanism to recapture donations send to the associated `SwapFacility` vaults by the owner. This mechanism can also be abused by a griefer in a frontrun to exhaust the `SwapFacility` buffer or `preMintedDebtTarget` value. Exhaustion of this buffer, would yield a number of functions inoperable and effectively in a Denial-of-Service state until some remediation, including `swapExactCollateralForDebt, swapExactCollateralForDebtAndLZSend, preMintDebt, burnExcessDebt`.

The scenario could occur where an EOA authorized to call `collectFees()` submits such a transaction to the mempool. The griefing party sees and frontruns it, by donating the buffer amount directly to the associated `SwapFacility Vault`, which for example we can assume is 1 WBTC. When the `collectFees()` transaction confirms, it will pull any fees out, and the buffer/1 WBTC donation. The buffer is now exhausted, and will stay so, until some parties begin swapping in debt for collateral. The buffer will stay at least partially depleted until a debt swap of the donation/buffer occurs, changing the expected behaviour of the contract.

Realistically, the `feeRecipient` could quickly remediate by swapping the collected donation + fee (fee amt not required) amount into `swapDebtForExactCollateral`.

Due to the ease of remediation, this is unlikely to be economically worthwhile for an attacker unless the buffer is low in economic value, thereby reducing the economic cost of this attack. Although one strength of this attack is it could even be done on the contract while it's in a paused state.

Recommendation: A proper sequence of calls can eliminate the potential for this attack to exhaust the buffer, even temporarily, thus avoiding DoS. A multicall with `preMintDebt()` prior to `collectFees()` is required. Due to the fact that `preMintDebt()` may revert in a number of scenarios, a multicall variant that continues to attempt followup calls following reverts would need to be utilized. Another option would be to hardcode the call to `preMintDebt()` via `collectFees()` before any calculation.

Bitcorn: Mitigated in new fee design where fees are explicitly managed in storage variable, see commit ab24673b.

Cantina Managed: Fixed.

3.4.5 Incorrect pause implementation results in confusing error messages in `BitcornOFT`

Severity: Low Risk

Context: BitcornOFT.sol#L98

Description: In `BitcornOFT.sol`, the `_beforeTokenTransfer` hook implements pause functionality by adding a `whenNotPaused` modifier while inheriting `ERC20PausableUpgradeable`. This results in the following two issues:

- The `whenNotPaused` modifier check occurs before the `ERC20PausableUpgradeable` implementation's check.
- The `ERC20PausableUpgradeable's` more descriptive error message is never reached.

As a result, users receive a generic `Pausable: paused` error instead of the more specific `ERC20Pausable: token transfer while paused`.

Recommendation: Consider removing the `whenNotPaused` modifier as the pause check is already handled by `ERC20PausableUpgradeable`

Bitcorn: Fixed in commit d17325af

Cantina Managed: Verified fix.

3.4.6 Lack of null check for `_sendParam.to` can result in crediting tokens crosschain to 0xdead address, resulting in loss

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `BitcornOFT` & `WrappedBitcoinNativeOFTAdapter` support crediting tokens to null address. In the case of `BitcornOFT` it works around the blocking of minting to `address(0)` by redirecting the minting to `address(0xdead)`. As the adapter does native transfers, it can send to `address(0)` normally.

The lack of null check here on the send parameters, on both ends, could easily result either from a bug or a mistake in user's funds ending up locked from a crosschain LZ transfer.

Recommendation: Enforce a null check on the `_sendParam.to` parameter to ensure these transfers cannot happen.

Bitcorn: Acknowledged, maintaining existing behavior of OFT contracts despite the pointed out inconsistency.

Cantina Managed: Acknowledged.

3.4.7 `SwapInFeeRate` and `SwapOutFeeRate` are applied differently

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In `SwapFacility`, the swap in fee is applied as a percentage to withhold on the collateral coming in. Meaning if the fee is 100% there is a full clawback, and the user receives no `debtTokens`.

The swap out fee, on the other hand is applied as a surcharge. Even if set to 100%, there isn't any actual clawback that occurs, and instead a user has to enter debt-to-collateral at a 2:1 ratio, or double the price. This methodology is somewhat unintuitive, compared to the former, but it has the benefit of ensuring that users always get something, and that a clawback isn't possible (albeit this should be enforced via saner fee limits).

For the `feeRecipient`, this isn't a particular issue, as they will receive the same fee amount on either side of the swap, for the same collateral amounts, however, it could be quite perplexing to users that fees are charged in different methodologies across the swaps, and makes it hard to reason about.

Recommendation: Decide on a consistent methodology for applying the fee to achieve the same fee capture style across swap types. The former is likely the most intuitive, in which case the withholding would be applied on the debt, rather than as a surcharge. This would require changing the `function swapDebtForExactCollateral(uint256 collateralOut, uint256 debtInMax, address to, uint256`

`deadline)` to something closer to `function swapExactDebtForCollateral(uint256 debtIn, uint256 collateralOutMin, address to, uint256 deadline)` which would also align it closer to the behaviour of `function swapExactCollateralForDebt(uint256 collateralIn, uint256 debtOutMin, address to, uint256 deadline)`

This makes it quite simple to precalculate and reason about the minimum amounts in this case, as they should just be `(inAmt * (1 - swapRate))` in either case.

Bitcorn: This was applied as a a multifaceted change without a specific commit in SwapFacility.sol#L182.

Cantina Managed: Verified fix.

3.5 Gas Optimization

3.5.1 Cache state variables with multiple accesses within a scope

Severity: Gas Optimization

Context: SwapFacility.sol#L307, SwapFacility.sol#L317-L318

Description: Gas savings can be had by reducing SLOAD operations, which is one of the most expensive operations on the EVM. Specifically when accessing the same state variable within a scope.

In one of the the referenced code blocks, `debtMinted` is cached into `currentDebt` but `debtMintCap` is not, while both have multiple accesses.

Recommendation: If a state variable is accessed more than once within a scope, cache it into a local variable, so rather than depending on an SLOAD for subsequent accesses, it's on the stack, saving significant gas potentially, the more it is accessed. Cache instances of `debtMintCap` & `debtMinted`.

Bitcorn: Mitigated in commit 6ec1042a.

Cantina Managed: Verified fix.

3.6 Informational

3.6.1 Calls to `SwapFacility._calcFee()` with `_feeRate` of 0 will revert, but should be handled

Severity: Informational

Context: SwapFacility.sol#L293

Description: If `_feeRate` were to be 0, and this codepath reachable, contract would revert on any codepaths that attempt this fee calculation, thereby allowing owners to partially pause the affected functions. The contract in its current state is not affected thanks to the preconditional checks prior to its 2 calls that `feeRate` is greater than 0, but as it is upgradeable, future code changes may miss adding that necessary step.

Recommendation: It is recommended to have the non-zero fee rate check within the function itself, and return a fee of 0 early in a null case, skipping the calculation. This will additionally remove the need for duplicated checks of this with preconditions in the various swap codepaths.

Bitcorn: Mitigated by adding early return with `feeRate = 0` in commit bef838b2.

Cantina Managed: Verified fix.

3.6.2 Restrict visibility specifier to lowest necessary visibility

Severity: Informational

Context: SwapFacility.sol#L255, SwapFacility.sol#L378, SwapFacility.sol#L386

Description: A number of functions have the `public` visibility whilst not having any associated internal references to the functions.

Recommendation: Since they are unused internally, it would be ideal to restrict them to `external` and make their usage clearer.

Bitcorn: Mitigated for `setSwapInFeeRate` and `setSwapOutFeeRate` in commit fa64e280. Note that `getDebtInForExactCollateral` was removed as part of broader change in fee processing.

Cantina Managed: Verified fix.

3.6.3 Minor code quality issues: typos

Severity: Informational

Context: SwapFacility.sol#L34, SwapFacility.sol#L62

Description: The codebase contains minor typos and naming inconsistencies that should be addressed for better code quality and maintainability.

- `SwapFacility.sol`

```
  - /// @dev collateralToken must have more > two and <= than 18 decimals
  + /// @dev collateralToken must have >= 8 and <= 18 decimals

  - /// @dev key trust assumption: this contract must be able to freely transfer collateralToken on
↪  behalf of the vault and only the vault.s
  + /// @dev key trust assumption: this contract must be able to freely transfer collateralToken on
↪  behalf of the vault and only the vault
```

Recommendation: Consider fixing the typos and naming inconsistencies to improve code quality.

Bitcorn: Fixed in commit 8e44fa5d.

Cantina Managed: Verified fix.

Bitcorn: Mitigated in commit 8e44fa5d. Spellchecker was however not added to IDE.

Cantina Managed: Verified fix.

3.6.4 Remove unused file imports

Severity: Informational

Context: BitcornOFT.sol#L18, WrappedBitcornNativeOFTAdapter.sol#L11-L13

Description: Multiple files across the entire repository contain an import statement that is not used anywhere in the contract, affecting code quality.

Recommendation: Consider removing unused file imports.

Bitcorn: Fixed in commits fb11c308 and 7353f960.

Cantina Managed: Verified fix. The unused imports are now removed.

3.6.5 Function `sharedDecimals` incorrectly marked as `view` instead of `pure`

Severity: Informational

Context: BitcornOFT.sol#L183

Description: In `BitcornOFT.sol`, the `sharedDecimals` function is declared as view but returns a constant value of 8. Since this function doesn't read any state variables, it should be marked as pure.

The same function is used in `WrappedBitcornNativeOFTAdapter.sol` but is correctly marked as `pure`, showing inconsistency across the codebase.

Recommendation: Change the function modifier from view to pure:

```
function sharedDecimals() public pure override returns (uint8) {
    return 8;
}
```

Bitcorn: Fixed in commit 5721dcc4.

Cantina Managed: Verified fix.

3.6.6 `ERC20Pausable` is initialized twice in `WrappedBitcornNativeOFTAdapter`

Severity: Informational

Context: WrappedBitcornNativeOFTAdapter.sol#L61

Description: The `initialize` function in `WrappedBitcornNativeOFTAdapter.sol` calls `__ERC20Pausable_init()` twice, which is unnecessary and indicates a code quality issue.

Recommendation: Remove one of the duplicate calls:

```
function initialize(address initialAuthority, address _delegate) public initializer {
    __Ownable_init(_delegate);
    __OFTCore_init(_delegate);
    __ERC20_init("Wrapped Bitcorn OFT", "WBTCN");
    __ERC20Pausable_init();  // Keep only one call
    __ERC20Permit_init("Wrapped Bitcorn OFT");
    _initializeAuthority(initialAuthority);
    __ReentrancyGuard_init_unchained();
}
```

Bitcorn: Fixed in commit 83d0dcbc.

Cantina Managed: Verified fix.

3.6.7 Remove unused `_delegate` parameter from `BitcornOFT` constructor

Severity: Informational

Context: BitcornOFT.sol#L31

Description: The `BitcornOFT.sol` contract's constructor includes an unused `_delegate` parameter that is never utilized in the constructor body:

```
constructor(address _lzEndpoint, address /*_delegate*/ )
    OFTCore(18, _lzEndpoint, address(this))
    Ownable(address(this))
{
    _disableInitializers();
}
```

Recommendation: Remove the unused parameter from the constructor.

Bitcorn: Fixed in commit deb33ab2.

Cantina Managed: Verified fix.

3.6.8 Consider removing `whenNotPaused` on permissioned functions, to aid in maintenance scenarios

Severity: Informational

Context: SwapFacility.sol#L378, SwapFacility.sol#L386, SwapFacility.sol#L404, SwapFacility.sol#L411, SwapFacility.sol#L418, SwapFacility.sol#L425, SwapFacility.sol#L432, SwapFacility.sol#L439

Description: A number of functions that are already permissioned, and not accessible without the express intention of the owners, are also protected by `whenNotPaused`. There are likely scenarios where the owner may wish to change some of the permissioned settings while the contract is in a paused state, and requiring unpausing could impede maintenance or recovery on the contracts.

Recommendation: Consider removing `whenNotPaused` to give the owners a safer and more flexible avenue to access permissioned functions, such as in a paused state.

Bitcorn: Mitigated in commit e11d0061.

Cantina Managed: Verified fix.

3.6.9 Redundant fee initialization in `_swapExactCollateralForDebt` function

Severity: Informational

Context: SwapFacility.sol#L266

Description: In `SwapFacility.sol`, the `_swapExactCollateralForDebt` function unnecessarily initializes the fee variable to 0, which is already the default value for uint256:

```
function _swapExactCollateralForDebt(uint256 collateralIn, uint256 feeRate)
    internal
    view
    returns (uint256 debtOut, uint256 fee)
{
    uint256 collateralIn18Decimals = collateralIn * to18ConversionFactor;

    fee = 0;  // Redundant initialization

    if (feeRate > 0) {
        fee = _calcFee(collateralIn18Decimals, feeRate);
        debtOut = collateralIn18Decimals - fee;
    } else {
        debtOut = collateralIn18Decimals;
    }
}
```

Recommendation: Consider removing the redundant initialization to save marginal gas costs.

Bitcorn: Fixed in commit bfb4f737

Cantina Managed: Verified fix.

3.6.10 Utilizing higher `sharedDecimals` values between OFTs leads to less dust accumulation for lowering LZ transfer bound

Severity: Informational

Context: BitcornOFT.sol#L32, BitcornOFT.sol#L177-L185, WrappedBitcornNativeOFTAdapter.sol#L48, WrappedBitcornNativeOFTAdapter.sol#L204-L212

Description: The `BitcornOFT` contract on L1 and `WrappedBitcornNativeOFTAdapter` contract on L2 are utilizing a `sharedDecimals` value of 8, attempting to match the Satoshi sub-division, whilst both utilize 18 decimals, including the native L2 version. When one Omnichain Fungible Token attempts transfers and has a lower `sharedDecimals` value than it uses, dust amounts below `sharedDecimals` are not sent and accumulate on that chain until they at least exceed that decimal amount.

Recommendation: Since both connected OFT tokens actually use 18 decimals, and `sharedDecimals` are intended to be set to the lowest common denominator, which would be 18 in this case, and save any dust issues from occurring on either chain.

However, as the client has noted, there is a `uint64` limit to consider for the LZ transfer, in which case the max individual transfer would be limited to ~18 BTC ($(2^{64}-1)/10^{18}$). It may be worthwhile to consider balancing the `sharedDecimals` value with the transfer bound, where the transfer bound at least meets the expected pre mint buffer, to minimize potential dust accumulation occurring, albeit the current USD value of one Satoshi is $0.000951, so the economic impact is almost negligible at current prices with the current settings.

Bitcorn: Acknowledged, retaining existing 8 `sharedDecimals` as per conversations about future non-EVM blockchain support.

Cantina Managed: Acknowledged.

3.6.11 `WrappedBitcornNativeOFTAdapter.send` could have sub-Satoshi minting side effect, due to lower `sharedDecimals`

Severity: Informational

Context: WrappedBitcornNativeOFTAdapter.sol#L154-L193

Description: `WrappedBitcornNativeOFTAdapter.send` function could be utilized inadvertently or purposely as a sub satoshi minter, due to `sharedDecimals` being set to 8, instead of the native 18, and in the case the `amountLD` parameter were denoted with a precision over 8 decimals. There's no negative or exploitative avenues found from this, as the additional `msg.value` used for the minting would remain in the adapter contract, and the `msg.sender` would be credited with the sub satoshi amounts on L2. It is, however, counter intuitive in that a user's balance following this call, could end up exceeding their prior balance subtracted by `amountSentLD`.

Recommendation: In case of a lower `sharedDecimal` value than decimals actually utilized for the OFT on the chain, consider precomputing `_debitView` with the dust removal, and then doing these calculations to account exactly how much `msg.value` should be minted is the realizable send amount, rather than requested send parameters.

Also fixed by setting `sharedDecimals` to native.

Bitcorn: Acknowledging the issue and potential for confusion. Retaining existing implementation.

Cantina Managed: Acknowledged.