Bitcorn OFT

Security Review

Cantina Managed review by:

R0bert, Lead Security Researcher
Om Parikh, Security Researcher

July 28, 2025

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 Jul 13th to Jul 16th the Cantina team conducted a review of bitcorn-oft on commit hash 46821811. The team identified a total of **14** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	2	0
Gas Optimizations	1	0	1
Informational	10	0	10
Total	**14**	**3**	**11**

3 Findings

3.1 Medium Risk

3.1.1 Depositing without checking health of underlying vault provides exit liquidity incase of shortfall / bad-debt

Severity: Medium Risk

Context: MorphoModule.sol#L75

Description: In the `deposit` function of the `MorphoModule` contract, WBTCN is borrowed from the `BitcornOFTAdapter` and deposited into a `MetaMorpho` vault (e.g., the BBQ BTCN vault) without performing any pre-deposit health checks on the vault or its underlying Morpho Blue markets:

```
function deposit(uint256 _amount) external requiresAuth whenNotPaused nonReentrant returns (uint256 _shares) {
    // Borrow WBTCN _amount from OFT
    BITCORN_OFT_ADAPTER.borrowWrapped(_amount);

    // Send the WBTCN to the MetaMorpho
    _shares = metaMorpho.deposit(_amount, address(this));
    emit Deposit(_amount, _shares);
}
```

This unconditionally calls `metaMorpho.deposit(_amount, address(this))`, which mints vault shares to the `MorphoModule` in exchange for the deposited assets. However, if the vault has unrealized or pending bad debt (e.g., from insolvent borrowers in attached Morpho Blue markets), the deposited WBTCN can inadvertently serve as "exit liquidity" for existing liquidity providers (LPs) attempting to withdraw. Because of this, there is no guarentee that the `MorphoModule` can later on withdraw all his shares since the underlying markets attached to vault are not fully backed.

Recommendation:

- Compare `lostAssets()` to the actual LP balance of the `address(1)` to determine the current bad debt amount.
- Add a governance-settable `uint256 public badDebtThreshold;` (in absolute wei or a percentage of `totalAssets()`). After the deposit:

```
if (lostAssets() > metaMorpho.convertToAssets(metaMorpho.balanceOf(address(1))) + badDebtThreshold) revert
↪  UnhealthyVault();
```

This way the deposit will revert if the bad debt of the underlying MetaMorpho Vault exceeds the `badDebtThreshold`.

Bitcorn: Fixed in PR #111

Cantina Managed: Fix verified. The `_checkBadDebt` is not needed/recommended in `collectYield` as Metamorpho v1.1 share price is non-reducing. To claim yield during bad-debt, an atomic operation of increasing the threshold, followed by collecting yield and restoring the threshold, would have to be executed.

3.2 Low Risk

3.2.1 Repayment operations in MorphoModule will revert if BitcornOFTAdapter is paused

Severity: Low Risk

Context: MorphoModule.sol#L89

Description: The `MorphoModule` contract facilitates automated borrowing of WBTCN tokens from the `BitcornOFTAdapter` and deposits them into a `MetaMorpho` vault for yield generation. In the `withdraw(uint256 _amount)` function, after withdrawing assets from the vault, the module attempts to repay the borrowed WBTCN via a call to `BITCORN_OFT_ADAPTER.repayWrapped(_amount)`. However, the `BitcornOFTAdapter` contract is pausable and its `_repayWrapped` internal function called by `repayWrapped` is indirectly guarded by the `whenNotPaused` modifier. More critically, if the adapter is paused, any repayment would be blocked.

Therefore, when the `BitcornOFTAdapter` is paused, the `MorphoModule`'s repay call will revert, preventing the module from closing its loan position. This is problematic because:

- Borrows in the system are uncollateralized.
- Pausing the adapter (e.g., for emergency or maintenance) would inadvertently lock the `MorphoModule`'s ability to repay.
- If the vault accrues yield or needs liquidation/rotation, the module can't unwind positions, leading to stuck funds or missed opportunities.

Essentially, if the adapter is paused to mitigate an issue (e.g., exploit in bridging), the `MorphoModule` would remain exposed with open borrows.

Recommendation: Consider removing the `whenNotPaused` modifier from the `_repayWrapped` function.

Bitcorn: Fixed in commit a29178f.

Cantina Managed: Fix verified.

3.2.2 Missing gaps (or any kind of storage extension) in upgradeable contracts

Severity: Low Risk

Context: MorphoModule.sol#L52

Description: `MorphoModule.sol` inherits from `UUPSUpgradeable` but lacks storage gaps. This limits the ability to add new storage variables in future upgrades without risking storage collisions or overrides.

Recommendation:

```
/**
 * @dev This empty reserved space is put in place to allow future versions to add new
 * variables without shifting down storage in the inheritance chain.
 * See https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
 */
uint256[50] private __gap;
```

Bitcorn: Fixed in commits:

- 2b0c1f2.
- f3abed4.
- d8f8591.

Cantina Managed: Fix verified. The size of the `__gap` array is usually calculated so that the amount of storage used by the contract always adds up to the same number (in this case 50 storage slots) and this was not respected but it does not suppose any security concern.

3.3 Gas Optimization

3.3.1 Low level call to transfer BTCN can be removed from the FlashloanModule

Severity: Gas Optimization

Context: FlashLoanModule.sol#L140-L142

Description: In the FlashLoanModule, the `_flashLoanBTCN` function transfers native BTCN via a low-level `.call{value: _amount}('')` without calldata, relying on the receiver's fallback/receive to accept it:

```
(bool success,) = address(_receiver).call{value: _amount}('');
if (!success) revert FlashLoanModule_TransferFailed();
```

If the receiving contract does not implement a receive/fallback function the flashloan call will revert.

Recommendation: Replace the low-level `.call` and simply forward the native BTCN as `value` in the `onFlashloan` call:

```
_receiver.onFlashLoan{value: _amount}(msg.sender, WBTCN, _amount, _fee, _data)
```

Update documentation to note the new receiver requirements.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4 Informational

3.4.1 A high flashloan fee in `FlashLoanModule` could reduce utility and competitiveness

Severity: Informational

Context: IntegrationBase.sol#L68

Description: The `FlashLoanModule` contract implements flashloans for BTCN and WBTCN tokens, borrowing liquidity from the `BitcornOFTAdapter` and charging a configurable fee (set via `setFlashLoanFee`). In the deployment and testing configuration (as seen in `IntegrationBase.sol`), the fee is initialized to 100 basis points (1%), calculated as `(_amount * flashLoanFee) / FEE_PRECISION` where `FEE_PRECISION = 10_000`.

```
// From test/modules/integration/IntegrationBase.sol:68
_flashloanFee = 100; // 1%
```

This fee structure could pose some issues in terms of utility and adoption:

1. Competition from fee-free alternatives in Morpho: The `MorphoModule` deposits borrowed WBTCN into a MetaMorpho vault (e.g., `BBQ_BTCN_MORPHO_VAULT_ADDRESS` at `0xa7Ba08CFc37e7CC67404d4996FFBB3E977490115`), which integrates with Morpho Blue. Morpho Blue supports flashloans with zero fees, limited only by available liquidity in the pool. As WBTCN liquidity grows in Morpho (driven by `MorphoModule` deposits), users can execute equivalent flashloans directly via Morpho without any cost. This diminishes the `FlashLoanModule`'s value proposition, especially since the module's liquidity is indirectly sourced from the same adapter and could be bypassed for cost savings.
2. High fee Relative to other protocols: A 1% fee is significantly higher than prevailing rates in established protocols: Aave (V3) charges ~0.09% for flash loans (9 basis points), making it a low-cost option for arbitrage, liquidations or collateral swaps and Balancer and other DEX-integrated lenders often charge 0.05-0.1%. Even in high-risk or niche markets, fees rarely exceed 0.5%. At 1%, the module becomes uncompetitive for large-volume users (e.g., a 1M USD flashloan costs 10K USD), potentially limiting adoption to scenarios where Morpho liquidity is insufficient or unavailable.
3. Higher Morpho liquidity (from module deposits) ironically reduces the `FlashLoanModule`'s relevance, creating a self-undermining dynamic. Users may prefer direct Morpho interactions, bypassing the module and its fee entirely.

No immediate security risks, but this could lead to underutilization of the `FlashLoanModule`.

Recommendation: Consider reducing the flash loan fee to a competitive level (e.g., 0.09-0.1%) via `setFlashLoanFee` (authorized by highsec roles) to align with Aave/Morpho standards and encourage usage. Differentiate the module with unique features, like bundled operations or lower gas via optimizations. Monitor Morpho liquidity post-deployment and adjust dynamically. In tests/deployment scripts, update the initializer to a lower default (e.g., `_flashloanFee = 9; // 0.09%`).

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.2 Insufficient timelock delay

Severity: Informational

Context: MainnetConstants.sol#L58

Description: The `TimelockController` contract (at address `0xaD2Bef31Db723b8ad1B9BCa41b0F1EBAfD1193d1`, as defined in `MainnetConstants.sol`) is assigned critical roles during deployment, including `HIGHSEC_OPERATIONS` and `UPGRADER`. These roles enable it to perform sensitive actions such as upgrading proxies (via `UUPSUpgradeable.upgradeToAndCall`), setting role capabilities (`RolesAuthority.setRoleCapability`), renaming roles (`Governor.setRoleName`) and assigning/revoking user roles (`RolesAuthority.setUserRole`). This setup is executed via batched transactions in the deployment scripts (e.g., `ExtensibleMinterSubmission.s.sol` and `ExtensibleMinterExecution.s.sol`), where the timelock gains these privileges to facilitate the initial upgrade and configuration of the `BitcornOFTAdapter` and modules.

However, the timelock's minimum delay (`minDelay`) is configured to only 60 seconds. This short delay could allow the quick execution of a potential malicious or erroneous proposal as with just 60 seconds

between queuing and execution, there is insufficient time for community review, audits, or alerts. An attacker compromising a multisig key (e.g., `CORN_ADMIN_MULTISIG`) could queue and execute harmful actions, such as upgrading to malicious implementations, revoking critical roles, or draining borrow caps, before detection or intervention.

Recommendation: Consider queuing critical proposals with a delay of 24 hours to achieve a better balance between operational efficiency and security oversight. Additionally, invoke `TimelockController.updateDelay` to increase `minDelay` to a higher value, ensuring sufficient time for thorough review and response in future governance actions.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.3 EOA address is given the unpause role during deployment

Severity: Informational

Context: DeploymentScript.t.sol#L40

Description: In the deployment and testing configuration (as defined in `DeploymentScript.t.sol`), the list of addresses granted `unpauser` permissions includes a mix of multisig wallets and a externally owned account (EOA). Specifically, the `_unpausers` array pushes three addresses:

```
// From test/modules/integration/DeploymentScript.t.sol:40-42
_unpausers.push(PAUSER_0);       // Not a multisig (EOA)
_unpausers.push(CORN_ADMIN_MULTISIG); // Multisig
_unpausers.push(UNPAUSER_2);     // Multisig
```

These unpausers are assigned roles via governance batches (e.g., in `ExtensibleMinterSubmission.s.sol` and `TimelockDataBatcher.sol`) to call `unpause()` on critical contracts like `BitcornOFTAdapter`, `MorphoModule` and `FlashLoanModule`. Unpausing restores normal operations (e.g., transfers, borrows, bridging) after an emergency pause, making it a high-privilege action.

While multisigs (e.g., `CORN_ADMIN_MULTISIG` and `UNPAUSER_2`) enforce distributed control through multi-signature requirements, mitigating the risk of single-key compromises, `PAUSER_0` operates as an EOA reliant on a solitary private key. Although this does not present an immediate exploit path (as it necessitates key compromise to activate), it decrements the robustness of the governance and emergency response framework, particularly in a protocol managing uncollateralized borrowing and cross-chain bridging operations.

Recommendation: Replace all single-signer EOAs (e.g., `PAUSER_0`) with multisig wallets in the `unpauser` list to ensure distributed control for critical recovery actions. Update deployment scripts (e.g., `DeploymentScript.t.sol`) to exclude EOAs and add only verified multisigs. For production, enforce a policy requiring at least 2/3 or similar thresholds for unpausing.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.4 Consider using a percent based borrow cap

Severity: Informational

Context: BitcornOFTAdapter.sol#L117-L122

Description: The `BitcornOFTAdapter` acts as a central liquidity provider for native BTCN (handled as ETH-like via `receive()` and `deposit()`) and wrapped WBTCN, supporting cross-chain bridging (via LayerZero OFT) and uncollateralized borrowing by trusted modules like `MorphoModule`. Borrowing operations draw directly from the adapter's native balance without explicit reservations for incoming bridge operations (`lzReceive`, which credits recipients via native BTCN transfers) or user `withdraw` calls.

In the borrowing flow (e.g., `borrowWrapped(uint256 _amount)` or variants), the adapter increases `borrowedBy` and `totalBorrowed` trackers, then deposits native BTCN into the WBTCN contract to mint and transfer wrapped tokens:

```
// From contracts/bitcorn/BitcornOFTAdapter.sol
function _creditWrapped(address _to, uint256 _amount) internal whenNotPaused {
    borrowedBy[_to] += _amount;
    totalBorrowed += _amount;

    // Deposit native tokens into WBTCN contract
    WBTCN.deposit{value: _amount}();

    // Transfer WBTCN tokens to the specified recipient
    WBTCN.safeTransfer(_to, _amount);
}
```

This depletes the adapter's native balance (`address(this).balance`) by `_amount` without reserving portions for:

- Incoming bridges (`lzReceive`): Credits unwrap native BTCN via `_to.call{value: _amountLD}('')`. If `balance < requested`, reverts `WithdrawalFailed`, stranding bridged funds (packet processed but credit fails).
- User withdrawals: `withdraw(uint256 _amount)` sends native BTCN via `msg.sender.call{value: _amount}('')`. Low level call reverts, preventing unwraps.

This can create a "liquidity trap": High borrowing (e.g., Morpho deposits to vaults) locks BTCN in external protocols, blocking bridge receives/withdraws until repayments. In worst cases, if borrowers (modules) can't repay (e.g., vault illiquidity), funds are stuck indefinitely.

Recommendation: Consider implementing a dynamic reserve mechanism in `BitcornOFTAdapter` to allocate a percentage of native balance for bridging/withdrawals. Introduce a `reserveRatio` (e.g., 50% = `5e17`, set via authorized `setReserveRatio(uint256)`), and adjust `globalAvailableToBorrow()` to enforce it:

```
// Proposed addition to BitcornOFTAdapter.sol
uint256 public reserveRatio;  // e.g., 5e17 for 50%, WAD-scaled

function setReserveRatio(uint256 _reserveRatio) external requiresAuth {
    if (_reserveRatio > WAD) revert InvalidReserveRatio();
    reserveRatio = _reserveRatio;
    emit ReserveRatioUpdated(_reserveRatio);
}

function globalAvailableToBorrow() public view override returns (uint256) {
    uint256 balance = address(this).balance;
    uint256 reserved = balance.mulDiv(reserveRatio, WAD);
    uint256 effectiveCap = balance - reserved;
    uint256 borrowed = totalBorrowed;
    return effectiveCap > borrowed ? effectiveCap - borrowed : 0;
}
```

Update `_checkBorrowingCaps` to use this implementation.

Bitcorn: Acknowledged. The recommended changes will not be adopted. This is because sensible limits will be placed on the borrow amounts.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.5 Bad debt can be ignored in yield collection

Severity: Informational

Context: MorphoModule.sol#L96-L98

Description: The `MorphoModule` integrates with a MetaMorpho v1.1 vault, where bad debt (losses from undercollateralized loans in Morpho Blue) is not automatically realized. Instead, it requires manual coverage via donations of assets to `address(1)` (a blackhole address), which are then used to offset losses before yield can be claimed.

In `collectYield()`, the module checks for unrealized bad debt by comparing `lostAssets` (vault's reported losses) against the converted assets from donations to `address(1)`:

```
// From contracts/modules/MorphoModule.sol:96-98
uint256 lostAssets = metaMorpho.lostAssets();
uint256 addressOneDeposits = metaMorpho.convertToAssets(metaMorpho.balanceOf(address(1)));
if (lostAssets > addressOneDeposits) revert LostAssetsExceedsDonationsToAddressOne();
```

If `lostAssets > addressOneDeposits`, the function reverts, preventing yield claims until sufficient donations are made to cover the shortfall. This creates delays in accessing surplus WBTCN (yield earned beyond borrowed amounts), which could be needed in some scenarios.

This safeguard is not necessary as unrealized bad debt does not affect the actual share price. At worst, the last users withdrawing from the vault will not be able to withdraw their full amount of shares as some shares will be unbacked because of the bad debt.

Recommendation: Consider removing the unrealized bad debt checks from the `collectYield()` function.

Bitcorn: Acknowledged. The recommended changes will not be adopted. While it is accurate to highlight the reliance on external manual intervention and the length of time this may take, this has been considered.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.6 Transient storage can be used for the `ReentrancyGuard`

Severity: Informational

Context: MorphoModule.sol#L26

Description: The library ReentrancyGuardUpgradeable is used. There is also a version that uses transient storage that is more efficient. Corn blockchain does support transient storage:

```
% cast call --rpc-url 'https://maizenet-rpc.usecorn.com' --create 0x5f5c
0x
```

Recommendation: Consider using ReentrancyGuardTransientUpgradeable.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.7 Current `RolesAuthority` permissions

Severity: Informational

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

Description: During the review, the following bash script was develop to capture the current role permissions assigned across the protocol (see gist d9441b28). These were the results:

```
% bash authorityRoles.sh
>>> Fetching on-chain role names via getRoleName()...
  [0] → admin
  [1] → bitcorn minter
  [2] → bitcorn minter with mintTo
  [3] → bitcorn burner
  [4] → bitcorn burner with burnFrom
  [5] → pauser
  [6] → unpauser
  [7] → operator lowsec
  [8] → operator highsec
  [9] → operator highsec timelocked
  [10] → upgrader
  [11] → swapFacility swapIn
  [12] → swapFacility swapOut
  [13] → create3 deploy user
  [14] → corn minter
  [15] → corn minter with mintTo
  [16] → corn burner
  [17] → corn burner with burnFrom

=== User Roles ===
```

```
  User: 0x095e7c378ae97b27bfa22016575e59c2267a14fc (Contract)
    Roles:
      0 : admin
      9 : operator highsec timelocked
      10 : upgrader
  User: 0x2029f88e5a98b90791da2c8982c8bb8f520a246c (Multisig(threshold=2))
    Roles:
      7 : operator lowsec
  User: 0x250d1567407d68ca6889c11a153a3c0e857954c8 (Multisig(threshold=2))
    Roles:
      0 : admin
      5 : pauser
      6 : unpauser
      7 : operator lowsec
      8 : operator highsec
      13 : create3 deploy user
  User: 0x76e1e3ba672a1c5804f9813168c1e8b114fb836c (EOA)
    Roles:
      5 : pauser
  User: 0x770a2350a9dc3daa77cdab2dd7f15aae2a44e8c1 (EOA)
    Roles:
      13 : create3 deploy user
  User: 0x84d986de22e12e6d959152117537b4feef3734dd (EOA)
    No roles.
  User: 0xaaad5ae4481a8cbc8dd2a0b5ee32f0b3222d337c (EOA)
    Roles:
      5 : pauser
  User: 0xbbb7cd886b3522f0b558d45a6fa2c5b8dab00ca8 (EOA)
    No roles.
  User: 0xc42879b3bbea96f32ce3cd67cd71f1c5335916a9 (EOA)
    No roles.
  User: 0xec013c81ee27b086966ba0b7cba87624e679b740 (EOA)
    Roles:
      5 : pauser
      6 : unpauser
      13 : create3 deploy user

=== Public Capabilities ===

=== Role-Based Capabilities ===
  Role 13  0x0969f8752a32b4f1f9d07b751c7bacbf6d9ae733 (Contract) 3ce98582
  ↪  deployWithCreationCodeAndConstructorArgs(string,bytes,bytes): GRANT
  Role 13  0x0969f8752a32b4f1f9d07b751c7bacbf6d9ae733 (Contract) 3ce98582
  ↪  deployWithCreationCodeAndConstructorArgs(string,bytes,bytes): GRANT
  Role 13  0x0969f8752a32b4f1f9d07b751c7bacbf6d9ae733 (Contract) 5ad63cab
  ↪  deployWithCreationCode(string,bytes): GRANT
  Role 13 → 0x0969f8752a32b4f1f9d07b751c7bacbf6d9ae733 (Contract) be3397e9 → deploy(string,bytes): GRANT
  Role 0 → 0x44f49ff0da2498bcb1d3dc7c0f999578f67fd8c6 (Contract) 3400288b → setPeer(uint32,bytes32): GRANT
  Role 0 → 0x44f49ff0da2498bcb1d3dc7c0f999578f67fd8c6 (Contract) ca5eb5e1 → setDelegate(address): GRANT
  Role 5 → 0x6e0bb5b7fff593859d9add6f50b46d50d87ec1fb (Contract) 8456cb59 → pause(): GRANT
  Role 6 → 0x6e0bb5b7fff593859d9add6f50b46d50d87ec1fb (Contract) 3f4ba83a → unpause(): GRANT
  Role 7 → 0x6e0bb5b7fff593859d9add6f50b46d50d87ec1fb (Contract) d4311d7f → UNKNOWN: GRANT
  Role 8  0x6e0bb5b7fff593859d9add6f50b46d50d87ec1fb (Contract) 2f940c70  emergencyWithdraw(uint256,address):
  ↪  GRANT
  Role 10  0x6e0bb5b7fff593859d9add6f50b46d50d87ec1fb (Contract) 4f1ef286  upgradeToAndCall(address,bytes):
  ↪  GRANT
  Role 9 → 0x6e0bb5b7fff593859d9add6f50b46d50d87ec1fb (Contract) 973b294f → setWithdrawalPeriod(uint256): GRANT
  Role 5 → 0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) 8456cb59 → pause(): GRANT
  Role 6 → 0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) 3f4ba83a → unpause(): GRANT
  Role 7 → 0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) d4311d7f → UNKNOWN: GRANT
  Role 8  0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) 2f940c70  emergencyWithdraw(uint256,address):
  ↪  GRANT
  Role 10  0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) 4f1ef286  upgradeToAndCall(address,bytes):
  ↪  GRANT
  Role 8 → 0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) a7ecd37e → updateSigner(address): GRANT
  Role 8  0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) 03c15957  updateMerkleRoot(bytes32,uint256):
  ↪  GRANT
  Role 8 → 0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) cd5b4920 → UNKNOWN: GRANT
  Role 8 → 0x2de76bbe5a785e7ed9af296097207f995d52af49 (Contract) 6bdbcfbf → UNKNOWN: GRANT
  Role 5 → 0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) 8456cb59 → pause(): GRANT
  Role 6 → 0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) 3f4ba83a → unpause(): GRANT
  Role 7 → 0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) d4311d7f → UNKNOWN: GRANT
  Role 8  0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) 2f940c70  emergencyWithdraw(uint256,address):
  ↪  GRANT
  Role 10  0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) 4f1ef286  upgradeToAndCall(address,bytes):
  ↪  GRANT
```

```
  Role 8 → 0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) a7ecd37e → updateSigner(address): GRANT
  Role 8  0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) 03c15957  updateMerkleRoot(bytes32,uint256):
  ↪  GRANT
  Role 8 → 0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) cd5b4920 → UNKNOWN: GRANT
  Role 8 → 0x514f0c7c0a4a925883689f283f3c2e8c1588b030 (Contract) 6bdbcfbf → UNKNOWN: GRANT
  Role 6 → 0x44f49ff0da2498bcb1d3dc7c0f999578f67fd8c6 (Contract) 3f4ba83a → unpause(): GRANT
  Role 5 → 0x44f49ff0da2498bcb1d3dc7c0f999578f67fd8c6 (Contract) 8456cb59 → pause(): GRANT

=== Burned Capabilities ===

Done.
```

This same script can be executed after the upgrade to ensure the new roles were assigned correctly.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.8 Typos, styling & code suggestions

Severity: Informational

Context: MorphoModule.sol#L39, ExtensibleMinterExecution.s.sol#L78

Description:

- Rename `IMetaMorpho` to `IMetaMorphoV1_1` (official name) to make it more verbose and explicit.
- `_oftTmplementation` is not spelled correctly, it should be `_oftImplementation`.

Bitcorn: Acknowledged. While the metaMorpho variable will not be modified, the `_oftTmplementation` variable is clearly a typo and was correct in the commit 69c2210

Cantina Managed: Partially solved by the Bitcorn team as the typo was corrected.

3.4.9 Avoid immutable variable with UUPS proxy pattern

Severity: Informational

Context: MorphoModule.sol#L36

Description: `BITCORN_OFT_ADAPTER` is declared as an `immutable` variable in the FlashLoanModule and MorphoModule contracts, which are implemented as UUPS proxies. This can be problematic in the following scenarios:

- When pointing multiple proxies to the same implementation, as the immutable value is baked into the bytecode and cannot vary per proxy instance.
- During frequent upgrades, as it adds operational overhead to ensure the value is correctly set in each new implementation's bytecode, rather than being managed as proxy state.

Recommendation:

- Consider making it a `constant` if the value is fixed and unlikely to change (e.g., for compile-time optimization while avoiding runtime storage reads).
- Alternatively, use a storage variable (e.g., initialized in the proxy's initializer) for flexibility, allowing updates via governance or upgrades without redeploying the implementation.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.

3.4.10 Deployment checklist & precautions

Severity: Informational

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

Description: The scripts in `script/extensibleMinting` are used for deploying and upgrading to the latest version. Based on the upgrade plan shared by the Bitcorn team, we recommend the following additions:

- Ensure the Foundry version is up to date.
- Ensure the Safe multisig version matches the one used in the `TransactionBatch` script.
- Ensure `package.json` and `pnpm-lock.yaml` are up to date and do not contain any malicious packages.
- Clean the build project to avoid any stale artifacts or caching issues.
- Generate the JSON multiple times with the same input parameters and compare the MD5 hash (or equivalent) of the file; this ensures file integrity is not compromised and that generation is idempotent and reproducible.
- Make sure all contracts are verified on the Corn explorer before approving and executing.
- Check the simulation trace of the Gnosis Safe transaction before approving and executing.

Bitcorn: Acknowledged.

Cantina Managed: Acknowledged by Bitcorn team.