Cornbase Bitcorn Security Review

Auditors

Deadrosesxyz, Lead Security Researcher

Kurt Barry, Lead Security Researcher

Jeiwan, Security Researcher

Sujith Somraaj, Associate Security Researcher

Report prepared by: Lucas Goiriz

May 15, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

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.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of cornbase according to the specific commit. Any modifications to the code will require a new security review.

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

3.1 Impact

- 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.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 10 days in total, Cornbase engaged with Spearbit to review the cornbase protocol. In this period of time a total of **23** issues were found.

Summary

Project Name	Cornbase
Repositories	corn-silo, bitcorn-token, silo-thorchain-aggregator
Commit	31dd9cf1, f51d0125, 6df30a0e
Type of Project	DeFi
Audit Timeline	Apr 22 to Apr 25
One day fix period	May 13 - May 14

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	11	8	3
Gas Optimizations	0	0	0
Informational	10	10	0
Total	**23**	**20**	**3**

5 Findings

5.1 Medium Risk

5.1.1 Tokens are bridged to `user` instead of `recipient` in `_approveAndBridgeBitcorn`

Severity: Medium Risk

Context: CornSilo.sol#L562

Description: The function `_approveAndBridgeBitcorn` in `CornSilo` bridges bitcoin tokens using the `ERC20Inbox` contract.

The `recipient` parameter is not used as intended in this function. The tokens are bridged to the `user` address instead of the `recipient` address.

```
erc20Inbox.createRetryableTicket(
    user, amount - (cost + maxGas * gasPriceBid), cost, user, user, maxGas, gasPriceBid, amount, ""
);
```

However, the `TokenBridged` event is emitted as if the tokens are bridged to the `recipient` address.

```
emit TokenBridged(address(bitcorn), user, recipient, amount, maxGas, gasPriceBid, data);
```

This inconsistency may lead to confusion and unexpected behavior, as the function caller might expect the Bitcorn tokens to be bridged to the specified `recipient` address. Still, instead, they are bridged to the `user` address.

Recommendation: Consider using the `recipient` parameter as intended in the `_approveAndBridgeBitcorn` function (or) remove it if redundant.

Bitcorn: Possible oversight due to lack of test coverage, fixed as suggested and a test has been added. Fixed in commits b7e49663 and 3c76a12d.

Spearbit: Fix verified.

5.1.2 BTCN cannot be used to pay gas when bridging approved assets to L2

Severity: Medium Risk

Context: CornSilo.sol#L538-L540

Description: The protocol implements the Bitcorn token (BTCN) that's used as the native coin on the Corn L2 network. The token is also used to pay bridging gas fees when sending tokens from Ethereum to the L2 network (when bridging Bitcorn, the gas fee is subtracted from the bridged amount). However, when bridging approved tokens, Bitcorn is not pulled to pay for the gas fee (CornSilo.sol#L537-L540):

```
IERC20Upgradeable(token).safeApprove(gatewayRouter.getGateway(token), amount);
IERC20Upgradeable(address(bitcorn)).safeApprove(gatewayRouter.getGateway(address(bitcorn)), btcnGas);

gatewayRouter.outboundTransferCustomRefund(token, recipient, recipient, amount, maxGas, gasPriceBid,
↪  data);
```

In the code snippet above, `token` is approved to its gateway, and `bitcorn` is approved to its gateway. Then, the token gateway is called (via the router) to pull the token and bridge it to the L2 network. However, there's no way for the token gateway to pull the gas fee in BTCN because Bitcorn is approved only to its gateway, but the Bitcorn gateway is not called.

As a result, either of the two will happen:

1. It won't be possible to bridge assets to L2 via `CornSilo` because the gateway will try to pull the gas fee and fail.
2. In the case of gas-less asset bridging from L1, it won't be possible to redeem tickets on L2 because redeeming requires gas to pay for the minting of tokens on L2.

Recommendation: In the `CornSilo._approveAndBridgeAsset()` function, consider approving Bitcorn to the token gateway, instead of approving it to the Bitcorn gateway. The gateway would then need to pull the gas fee in Bitcorn, as well as the token asset itself.

Bitcorn: In practice, the token gateway contract is the same for BTCN and other tokens so the bridge call still works. However, we switched the approval to the recommendation of using the same token address gateway to standardize the code in commit bf213b89.

Spearbit: Fix verified.

5.2 Low Risk

5.2.1 Users cannot specify `minAmountOut` when calling `depositDAI`

Severity: Low Risk

Context: CornSiloZap.sol#L272

Description: Since SDAI will be an approved token within `CornSilo`, if users wish to deposit `DAI`, they'll have to first deposit it into `SDAI`. The problem is that the `DAI`:`SDAI` ratio is not strictly 1:1 and can change. Currently, when using the `depositUSDC` function, users can specify the `minAmountOut` SDAI they'd like to receive. However, `depositDAI` lacks such functionality.

```
SDAI.deposit(receivedDai, address(this));
uint256 receivedSDai = SDAI.balanceOf(address(this)) - previousSDai;
require(receivedSDai >= minAmount, "Not enough received in swap");
```

Recommendation: Allow users to specify `minAmount` of SDAI that they're willing to receive.

Bitcorn: Fixed as suggested. minAmountOut parameter added that confirms sDAI amount actually recieved, in commit 5950f7c2. Additionally, the same fix is made for depositing DAI with permit pathway, in commit 578a6625.

Spearbit: Fix verified.

5.2.2 It should not be possible to add `Bitcorn` as an approved token

Severity: Low Risk

Context: CornSilo.sol#L764

Description: Within the contract, there's different logic when dealing with `approved` tokens and when dealing with `Bitcorn`. If `Bitcorn` is accidentally added as an approved token, it would allow using `bridgeApprovedToken` and `redeemApprovedToken` for `Bitcorn`. It would furthermore break `bridgeAllTokens` and `redeemAllTokens` as it would double-spend `Bitcorn`.

Recommendation: Change the `if-check` in `addApprovedToken` as follows:

```
- if (token == address(bitcornMinterAsset)) {
+ if (token == address(bitcornMinterAsset) || token == address(bitcorn)) {
```

Bitcorn: Fixed in commit a92bbec6 as suggested, with a slightly different logic to conform to the custom error message approach used in contract.

Spearbit: Fix verified.

5.2.3 `bridgeAllTokens` might unexpectedly revert if `remainingBTCNShares` is a non-zero value, less than `cost + maxGas * gasPriceBid`

Severity: Low Risk

Context: CornSilo.sol#L563

Description: Within `bridgeAllTokens`, after all approved tokens are bridged, if the user has any `Bitcorn` shares left, the contract attempts to bridge them too.

```
// Bridge BTCN if present, transferring backing assets to SwapFacility
if (remainingBTCNShares != 0) {
    _approveAndBridgeBitcorn(user, cost, remainingBTCNShares, recipient, maxGas, gasPriceBid, data);
    IERC20Upgradeable(address(bitcornMinterAsset)).safeTransfer(
        _swapFacilityVault, _from18DecimalsToAssetDecimals(remainingBTCNShares)
    );
}
```

The problem is that this `bridge` does not account for the Bitcorn gas costs and in case `remainingBTCNShares < cost + maxGas * gasPriceBid`, the following line within `_approveAndBridgeBitcorn` will revert due to underflow.

```
erc20Inbox.createRetryableTicket(
    user, amount - (cost + maxGas * gasPriceBid), cost, user, user, maxGas, gasPriceBid, amount, ""
);
```

Recommendation: Change the if-check to the following:

```
- if (remainingBTCNShares != 0) {
+ if (remainingBTCNShares != 0 && remainingBTCNShares  > cost + maxGas * gasPriceBid)) {
```

Bitcorn: Fixed in commit 17e9bd45 as suggested by adding a condition that `remainingBTCNShares > cost + maxGas * gasPriceBid` to allow continuation.

Spearbit: Fix verified.

5.2.4 Re-use of `data` param in `_bridgeAllTokens` function

Severity: Low Risk

Context: CornSilo.sol#L466

Description: In the `_bridgeAllTokens` function, the same data parameter is used to bridge both approved and Bitcorn (BTCN) tokens. This could lead to potential issues if the `data` parameter is intended to have different values for each type of token being bridged.

For bridging Bitcoin (BTCN) this value is overridden to `""` inside the `_approveAndBridgeBitcorn` function, however for all other approved token, the same `data` parameter is used.

```
function _bridgeAllTokens(
    address user,
    address recipient,
    uint256 cost,
    uint256 maxGas,
    uint256 gasPriceBid,
    bytes calldata data
) internal {
    // ...

    // Bridge all approved assets
    for (uint256 i = 0; i < approvedTokens.length; i++) {
        if (cachedShares[i] > 0) {
            _approveAndBridgeAsset(
                approvedTokens[i], user, cachedShares[i], gatewayRouter, recipient, maxGas,
↪   gasPriceBid, data
            );
        }
    }

    // ...

    // Bridge BTCN if present, transferring backing assets to SwapFacility
    if (remainingBTCNShares != 0) {
        _approveAndBridgeBitcorn(user, cost, remainingBTCNShares, recipient, maxGas, gasPriceBid, data);
        // ...
    }
}
```

Recommendation: Consider allowing users to specify different `data` parameters for bridging approved and Bitcorn tokens.

Bitcorn: Acknowledged. While this could lead to potential issues if the data parameter is intended to have different values for each type of token being bridged, each will be the same in practice. The changing to `""` in the case of BTCN is considered acceptable according to how that function works and can be explained better.

Spearbit: Acknowledged.

5.2.5 Inconsistency in data field usage between `createRetryableTicket` and `TokenBridged` event

Severity: Low Risk

Context: CornSilo.sol#L562-L566

Description: The function `_approveAndBridgeBitcorn` in `CornSilo.sol` bridges Bitcorn token from L1 -> L2 using the ERC20 Inbox contract. The `_approveAndBridgeBitcoin` function, in turn, uses the `createRetryableTicket` function to bridge Bitcorn tokens.

However, there needs to be more consistency in the usage of the data field between the `createRetryableTicket` function call and the `TokenBridged` event in the `_approveAndBridgeBitcorn` function.

In the `createRetryableTicket` function call, the data field is passed as an empty string `""`:

```
erc20Inbox.createRetryableTicket(
    user, amount - (cost + maxGas * gasPriceBid), cost, user, user, maxGas, gasPriceBid, amount, ""
);
```

However, in the `TokenBridged` event that follows, the data field is emitted with the value passed to the function:

```
emit TokenBridged(address(bitcoin), user, recipient, amount, maxGas, gasPriceBid, data);
```

Recommendation: Consider updating the `_approveAndBridgeBitcorn` function to use the same data field value in both the `createRetryableTicket` function call and the `TokenBridged` event (or) if the `data` field is redundant, remove it.

Bitcorn: Fixed in commit 5d59a6f5 as suggested. The data field was removed from the function signature. It is passed in as `""` to `createRetryableTicket` and this empty string is mirrored in event (the field in the event is preserved).

Spearbit: Fix verified.

5.2.6 Missing zero address check for `feeRecipient` if fees are greater than zero

Severity: Low Risk

Context: TSAggregator.sol#L29

Description: The `TSAggregator` contract allows privileged addresses to set the `_fee` and `_feeRecipient` through the `setFee()` function. The function checks if the fee set is within the allowed limit.

However, adding an extra defensive check is more logical to ensure the `_feeRecipient` is set to a valid address when the `_fee` values are not zero.

Recommendation: Consider adding a check to ensure `_feeRecipient` is not zero when the `_fee` values exceed zero.

```
  function setFee(uint256 _fee, address _feeRecipient) public isOwner {
      require(_fee <= MAX_FEE, "fee can not be more than 10%");
+     if(_fee > 0) require(_feeRecipient != address(0));
      fee = _fee;
      feeRecipient = _feeRecipient;
      emit FeeSet(_fee, _feeRecipient);
  }
```

Bitcorn: Fixed in commit 1756f34a as suggested.

Spearbit: Fix verified.

5.2.7 Changing swap fee in THORChain Aggregator can cause failing swaps

Severity: Low Risk

Context: TSAggregatorCornSiloEth.sol#L60

Description: The protocol integrates with THORChain via `TSAggregatorCornSiloEth` to allow users deposit their BTC into the `CornSilo` contract. Since THORChain converts bridged BTC only to ETH, an aggregator is required to swap ETH to WBTC and deposit it into `CornSilo`. The swapping step is performed in the `TSAggregatorCornSiloEth.swapOut()` and is done via the Uniswap V3 Router. To protect users against slippage, they're allowed to specify the minimum amount of WBTC to receive after the swap–this amount is calculated when the user is initiating bridging of BTC.

However, the aggregator takes a fee (TSAggregatorCornSiloEth.sol#L50-L51), which can impact the swap. If the fee is increased or set while there is a pending BTC deposit, the pre-computed minimal amount of WBTC will be higher than it should be. As a result, swapping may result in a lower-than-expected output amount of WBTC, which will cause a revert due to the slippage tolerance check in the Uniswap Router.

Recommendation: In the `TSAggregatorCornSiloEth.swapOut()` function, consider reducing `amountOutMin` by the fee percentage. This will also eliminate the need to account for the fee in the `amountOutMin` calculation when initiating bridging of BTC.

Bitcorn: Acknowledged. The fee feature is from the THORChain codebase and is not intended to be used in these aggregators. This will be noted for future governance.

Spearbit: Acknowledged.

5.2.8 Unrestricted Bitcorn minting via `CornSilo` can impact `SwapFacility`

Severity: Low Risk

Context: CornSilo.sol#L252, CornSilo.sol#L269-L274

Description: Unlike depositing of approved assets, which is allowed only until bridging is enabled, minting of Bitcorn tokens is allowed even after bridging is enabled. As per the documentation, this is done to let users mint BTCN and pay the bridging gas fee (CornSilo.sol#L249):

```
/// @dev Unlike approved tokens, BTCN can be deposited after the bridge is enabled to allow adding gas
↪  for bridge transfers.
```

However, this possibility is not limited by time. After bridging is enabled, all deposited WBTC will be transferred to the Swap Facility Vault (CornSilo.sol#L428-L430), which will serve as a MakerDAO's PSM-like facility: users will be able to mint more BTCN in exchange for WBTC. As per the documentation, a fixed fee will be paid by users when swapping via the Swap Facility Vault. However, since minting will also be enabled in the Corn Silo contract and there's no minting fee, users will tend to use the Silo contract to mint BTCN, instead of using the Swap Facility Vault, to avoid paying the fee.

As a result, the protocol will receive less fees from the Swap Facility Vault than expected; users will mostly use the Corn Silo contract to mint BTCN, instead of using the Swap Facility Vault.

Recommendation: Consider improving the token bridging functionality so that users can mint only the exact amount of BTCN required to bridge their tokens. This will require implementing alternative versions of the `CornSilo.bridgeToken()` and `CornSilo.bridgeAllTokens()` functions, where, instead of spending the user's Bitcorn shares, WBTC is pulled from the user to mint enough BTCN to pay for the bridging gas fee.

Bitcorn: Acknowledged. `SwapFacility` will maintain zero fees initially. BTCN minting on the `CornSilo` can be disabled by the governance before opening a `SwapFacility` with fee. From a user perspective, bridging is still possible for users with no BTCN as long as retriable ticket is redeemed on L2. Withdrawal of assets is always possible.

Spearbit: Acknowledged.

5.2.9 Wrong return values of `CornSilo.redeemAll()`

Severity: Low Risk

Context: CornSilo.sol#L608, CornSilo.sol#L699-L701

Description: As per the documentation, the `CornSilo.redeemAll()` function is expected to return:

```
/// @return depositedTokens array of tokens with shares found for the user
/// @return assets array of the underlying asset values returned to the user
```

However, it returns different values:

1. Instead of an array of tokens with shares found for the user (`depositedTokens`), it returns an array of all approved tokens: CornSilo.sol#L699.
2. Instead of an array of the underlying asset values returned to the user (`assets`) it returns an array of the amounts that were deposited by the user: CornSilo.sol#L700. As can be seen, the actual array of redeemed asset amounts is filled but not used and not returned: CornSilo.sol#L721.

As a result, when redeeming all tokens, users are forced to track their balances before and after redeeming to see how much assets were withdrawn. This contradicts the logic of `CornSilo.redeemToken()` and `CornSilo.redeemBitcorn()`, which return the actual redeemed amounts.

Recommendation: In the `CornSilo._redeemAllTokens()` function, consider returning:

1. `depositedAssets` instead of `approvedTokens` (the first returned value of the function).
2. `assetsReturned` instead of `depositedAssets` (the second returned value of the function).

Bitcorn: Fixed in commit 3a674aa0 as suggested with a further correction in 33317be5.

Spearbit: Fix verified.

5.2.10 `sweep()` may revert silently

Severity: Low Risk

Context: CornSiloZap.sol#L339

Description: The `sweep()` function is a privileged function that allows authorized actors to move stuck tokens, including native tokens, from the `CornSiloZap` contract to itself.

In this function, the contract transfers native tokens to the caller using the call function. However, it does not check the success status of the call operation. If the transfer fails for any reason (e.g., the recipient is a contract that doesn't accept Ether or throws an exception), the failure will be silently ignored. The function will continue executing as if the transfer was successful.

Recommendation: Consider checking the success status of the call operation and handle the failure appropriately.

```
  function sweep(IERC20 token) external requiresAuth {
     if (address(this).balance > 0) {
       (bool success,) = address(msg.sender).call{value: address(this).balance}("");
+      require(success, "ETH transfer failed");
      }
      // ...
  }
```

Bitcorn: Fixed in commit 172463f4 as suggested. Explicit success check added to ETH transfer with revert on failure.

Spearbit: Fix verified.

5.2.11 Missing zero address validation of authority

Severity: Low Risk

Context: Bitcorn.sol#L34, CornSilo.sol#L82, CornSiloZap.sol#L112

Description: When initializing the authority address in `Bitcorn`, `CornSilo`, and `CornSiloZap` contracts, the specified address is not validated. The `AuthNoOwner._initializeAuthority()` function also doesn't validate the specified address.

As a result, there's a risk of specifying the zero address as the authority address. In this case, it won't be possible to change the address or upgrade a contract and re-initialize it because there's no public setter and upgrading requires a valid authority contract (`_authorizeUpgrade()` functions are protected by the `requiresAuth` modifier). Thus, the only way to set a correct authority address would require redeploying a contract.

Recommendation: In the initializers of `Bitcorn`, `CornSilo`, and `CornSiloZap` contracts, consider ensuring that the passed `_authority` address is not zero.

Bitcorn: Fixed in commit 0b6cc912 for `CornSilo` and `CornSiloZap` as suggested. Authority validated to not be the zero address on initialization.

Spearbit: Fix for `CornSilo` and `CornSiloZap` confirmed; no fix provided for `Bitcorn`.

5.3 Informational

5.3.1 `TSAggregatorCornSiloEth.swapOut` does not send ETH if the swap fails, that is the responsibility of the calling contract

Severity: Informational

Context: TSAggregatorCornSiloEth.sol#L43

Description: This `@notice` comment on `TSAggregatorCornSiloEth.swapOut` says: "If the operation fails, user will receive ETH directly to their wallet.". This is not a behavior of `swapOut`, but rather the calling function in the `ThorchainRouter`. Note that `swapOut` could, in principle, be called directly, and this behavior would not occur.

Recommendation: Consider clarifying this comment refers not to a property of this function, but rather the expected behavior of the calling contract.

Bitcorn: Fixed in commit c0b1d009 as suggested. Comment clarified.

Spearbit: Fix verified.

5.3.2 Check in `_bridgeAllTokens` does not guarantee each maximum transfer fee is a multiple of one Satoshi, contrary to suggestion in comment

Severity: Informational

Context: CornSilo.sol#L496

Description: The `_bridgeAllTokens` function checks that the sum of all maximum gas costs is an even multiple of `ONE_SATOSHI_OF_BITCOIN_SHARES`:

```
uint256 requiredShares = maxGas * gasPriceBid * userApprovedTokenCount;
// ...
if (requiredShares % ONE_SATOSHI_OF_BITCOIN_SHARES != 0) {
    revert SharesNotMultipleOfOneSatoshi(requiredShares, ONE_SATOSHI_OF_BITCOIN_SHARES);
}
/// This should imply each individual transfer is also a clean multiple?
```

The comment suggests that this implies that each transfer also uses an even multiple of Satoshis for gas. However, it does not imply this. For WBTC, `ONE_SATOSHI_OF_BITCOIN_SHARES` is equal to 10^{10}. `maxGas * gasPriceBid * userApprovedTokenCount` may be divisible by 10^10 even if `maxGas * gasPriceBid` is not. For example, since $10^{10} = 2^{10} \cdot 5^{10}$, `maxGas * gasPriceBid` could be missing, say, a factor of 2, and then `userApprovedTokenCount` could be 2 or 4 or any other even number. E.g. `maxGas == 10**8`, `gasPriceBid == 50`, `userApprovedTokenCount == 2`. There is no clear negative consequence of this inconsistency, however.

Recommendation: Remove the inaccurate comment. If for some reason it is important that each individual transfer use an even amount of Satoshi in gas, modify the check to use `maxGas * gasPriceBid` instead of `requiredShares`.

Bitcorn: Fixed in commit ff3899f0 to make the comment true that the gas used for each individual `approvedToken` transfer is a clean multiple of `ONE_SATOSHI_OF_BITCOIN_SHARES`.

Spearbit: Fix verified.

5.3.3 Missing `ZeroWithdraw` check in `_redeemApprovedToken` function

Severity: Informational

Context: CornSilo.sol#L611

Description: The `_redeemApprovedToken` function allows users to redeem their shares of an approved token and withdraw the underlying assets. However, unlike `_redeemBitcornAndWithdrawMinterAsset`, a similar function that helps users redeem their Bitcorn shares, this function doesn't have a zero share check, allowing users to withdraw zero shares.

Recommendation: To improve code quality and consistency, consider adding a check to ensure the user is not redeeming zero shares.

```
function _redeemApprovedToken(address account, address token, uint256 shares) internal returns
↪  (uint256) {
    if (!_isApprovedToken(token)) {
        revert TokenNotApproved(token);
    }

    if (shares == 0) {
        revert ZeroWithdraw(address(bitcorn));
    }
    // ....
}
```

Bitcorn: Fixed in commit b711f234 as suggested.

Spearbit: Fix verified.

5.3.4 `CornSiloZap.depositUSDT` uses `balanceOf` for USDT received, which may not be accurate

Severity: Informational

Context: CornSiloZap.sol#L312

Description: Other deposit functions in this contract only attempt to deposit exactly the amount received from the user, which is determined either from the transfer amount, or by differencing pre- and post-balances. The exception is `depositUSDT`, which just uses the balance of the contract post-transfer. This amount may not be the received amount if USDT has been sent to the contract directly. The result is that the user may receive more funds than expected. This is certainly not a negative for the depositor; however, since the contract includes a recovery function for tokens it holds, it means that users who mistakenly send USDT directly to the contract likely cannot have their funds returned as the tokens they send will be taken by MEV bots or normal depositors before an admin can call `sweep()`.

Recommendation: Use the difference between pre- and post-USDT balances (as USDT may have its transfer fee activated some day) to determine the received USDT for consistency with other functions.

Bitcorn: Fixed in commit 797c73e3 to use the difference of pre-and-post operation USDT balances as suggested.

Spearbit: Fix verified.

5.3.5 Merge `fromAssetDecimalsTo18Decimals` and `_fromAssetDecimalsTo18Decimals` functions

Severity: Informational

Context: CornSilo.sol#L302

Description: The `CornSilo` contract has two functions that perform the same operation of converting an amount from the asset decimals to 18 decimals: `fromAssetDecimalsTo18Decimals` and `_fromAssetDecimalsTo18Decimals`.

The same functionality can be achieved by removing the `_fromAssetDecimalsTo18Decimals` internal function and converting `fromAssetDecimalsTo18Decimals` from an external to a public function.

Similar changes could be applied to the `from18DecimalsToAssetDecimals` function: CornSilo.sol#L309

Recommendation: Consider moving the calculation logic from `_fromAssetDecimalsTo18Decimals` to `fromAssetDecimalsTo18Decimals` and make it public. This will improve code quality and avoid code repetition.

```
- function fromAssetDecimalsTo18Decimals(uint256 amount) external view returns (uint256) {
-     return _fromAssetDecimalsTo18Decimals(amount);
- }
+ function fromAssetDecimalsTo18Decimals(uint256 amount) public view returns (uint256) {
+     return amount * to18ConversionFactor;
+ }
  // ...
- function _fromAssetDecimalsTo18Decimals(uint256 amount) internal view returns (uint256) {
-     return amount * to18ConversionFactor;
- }
```

Bitcorn: Modified in commit f7d74215 as suggested. Internal and external function variants consolidated to one public function.

Spearbit: Fix verified.

5.3.6 Use `receive()` instead of `fallback()` in `TSAggregator`

Severity: Informational

Context: TSAggregator.sol#L27

Description: The `TSAggregator` contract currently uses the `fallback()` function to receive native tokens (e.g., ETH) sent to the contract. However, the `fallback()` function is typically used when the transaction includes both native tokens and additional data (`msg.data`) that needs to be processed.

When the contract only needs to handle native token transfers without additional data, using the `receive()` function is more appropriate.

```
  // Needed for the swap router to be able to send back ETH
- fallback() external payable {}
+ receive() external payable {}
```

Recommendation: Consider replacing the `fallback()` function with `receive()` function to process incoming native token transfers.

Bitcorn: Fixed in commit 1756f34a as suggested.

Spearbit: Fix verified.

5.3.7 Declare `MAX_FEE` and `MAX_BPS` as constant

Severity: Informational

Context: TSAggregator.sol#L14-L15

Description: In the `TSAggregator` contract, `MAX_FEE` and `MAX_BPS` are declared state variables with fixed values.

Recommendation: Consider declaring them as constants.

```
- uint256 MAX_FEE = 1_000;
- uint256 MAX_FEE = 1_000;
+ uint256 private constant MAX_FEE = 1_000;
+ uint256 private constant MAX_BPS = 10_000;
```

Bitcorn: Fixed in commit 1756f34a as suggested.

Spearbit: Fix verified.

5.3.8 `IBitcorn` interface is incomplete

Severity: Informational

Context: IBitcorn.sol#L7

Description: The IBitcorn interface doesn't define all public and external functions implemented by the Bitcorn contract:

- Missing `IERC20MetadataUpgradeable` functions (implemented by `ERC20Upgradeable`).
- Missing `IERC20PermitUpgradeable` function (implemented by `ERC20PermitUpgradeable`).
- Missing `IERC1822ProxiableUpgradeable` and `IERC1967Upgradeable` function (implemented by `UUPSUpgradeable`).
- Missing `AuthNoOwner` functions, however there's no respective interface.

Recommendation: In the `IBitcorn` interface, consider defining all functions implemented by the Bitcorn contract.

Bitcorn: Partially fixed in commit 1e341457, and partially acknowledged:

- (Fixed) Token interfaces `IERC20MetadataUpgradeable` and `IERC20PermitUpgradeable` inherited in `IBitcorn`.
- (Acknowledged) Proxy interfaces `IERC1822ProxiableUpgradeable` and `IERC1967Upgradeable` not inherited.
  - We prefer to view these functions on a contract only when viewing as a proxy, which is consistent with other contracts using UUPS proxy pattern.
- (Acknowledged) `AuthNoOwner` functions not added as consistent with other inheriting contracts.

Spearbit: Fix verified.

5.3.9 Remove unused code

Severity: Informational

Context: ISwapFacility.sol#L4, CornSiloZap.sol#L5, CornSiloZap.sol#L15, TSAggregatorCornSiloEth.sol#L6, CornSiloZap.sol#L73, TSAggregator.sol#L19, TSAggregatorCornSiloEth.sol#L20, IBitcorn.sol#L20-L21, CornSiloZap.sol#L95-L102, CornSiloZap.sol#L333, CornSiloZap.sol#L88-L89, CornSiloZap.sol#L92, CornSiloZap.sol#L18

Description: Multiple files across the entire repository contain an import statement not used anywhere in the contract (or) its inheriting contracts.

Importing unused libraries can increase the contract's deployment and execution gas costs and make the codebase less readable and maintainable.

Moreover, there are other occurrences of unused code:

1. `swapFacility` is not used in `CornSiloZap`: CornSiloZap.sol#L73
2. `tokenTransferProxy` is not used in `TSAggregator`: TSAggregator.sol#L19
3. `FEE_TIER_500` is not used in `TSAggregatorCornSiloEth`: TSAggregatorCornSiloEth.sol#L20
4. `burnFrom` function still exists in the `IBitcorn` interface: IBitcorn.sol#L20-L21
5. `onlyEOA` modifier in `CornSiloZap`: CornSiloZap.sol#L95-L102
6. `_usdcTo18Decimals` function and its related constant variables in `CornSiloZap`: CornSiloZap.sol#L333
7. `WAD` in `CornSiloZap`: CornSiloZap.sol#L92
8. The `submit` function of the `ILido` interface in `CornSiloZap`: CornSiloZap.sol#L18

Recommendation: To optimize gas costs and improve code quality, consider removing the unused file imports from multiple contracts across the repository. Removing unused functions, constants, etc will improve the readability and maintainability of the codebase.

Bitcorn: Mostly fixed as suggested, and partially acknowledged:

- (Fixed in commit 903d0d12) `ISwapFacility` fixes `ISwapFacility.sol#L4`.
- (Fixed in commit cc15496b) `CornSiloZap` fixes `CornSiloZap.sol#L5`, `CornSiloZap.sol#L73`, `CornSiloZap.sol#L95-L102`, `CornSiloZap.sol#L333`, `CornSiloZap.sol#L88-L89`, `CornSiloZap.sol#L92`, `CornSiloZap.sol#L18`.
- (Fixed in commit 739a1080) `CornSiloZap` fixes `CornSiloZap.sol#L15`.
- (Fixed in commit 40948b47) `TSAggregatorCornSiloEth` fixes `TSAggregatorCornSiloEth.sol#L6`.
- (Acknowledged) `TSAggregatorCornSiloEth.sol#L20`, keeping for future iterations.
- (Acknowledged) `TSAggregator.sol#L19`, keeping to maintain functionality in this "library" contract.
- (Fixed in commit 4fe15fac) `IBitcorn` fixes `IBitcorn.sol#L20-L21`.

Spearbit: Fixes verified.

5.3.10 Minor code quality improvements

Severity: Informational

Context: CornSilo.sol#L246, CornSilo.sol#L262, CornSilo.sol#L191, CornSilo.sol#L218, CornSilo.sol#L592, CornSilo.sol#L688, CornSilo.sol#L585, CornSilo.sol#L192, CornSiloZap.sol#L271, CornSiloZap.sol#L288, TSAggregatorCornSiloEth.sol#L41

Description: The codebase contains minor typos and naming inconsistencies that can affect code quality and maintainability.

- `CornSilo.sol`:

```
  /**
  @dev Redemption of shares is subject with withdrawal fee if present // subject with -> subject
↪  to
  // ...
  @dev Approved assets are .... reciept to track their position // reciept -> receipt
  // ...
  @dev Approved assets are .... issued a non-transferrable reciept to track their position //
↪  reciept -> receipt
  // ...
  @dev The users non-transferrable reciept is stored as BTCN shares // reciept -> receipt
  // ...
  @dev The users non-transferrable reciept is stored as BTCN shares // reciept -> receipt
  // ...
  @notice Withdrawing ..... in point aquisition according to those mechanics // aquisition ->
↪  acquisition
  // ...
  @dev Convenience function ... No hard guarentee of executabilty ... // guarentee -> guarantee
↪  and executabilty -> executability
  // ...
  @dev because BTCN share mints are ... guarenteed to scale down to the minter asset cleanly. //
↪  guarenteed -> guaranteed
```

- `CornSiloZap.sol`

```
  /**
  // ...
  @param daiAmount The amount of DAI to depositl // depositl -> deposit
  // ...
  @param daiAmount The amount of DAI to depositl // depositl -> deposit
  // ...
```

- `TSAggregatorCornSiloEth.sol`

```
  /**
  // ...
  @notice Swap recieved ETH for WBTC and deposit into CornSilo on users behalf // recieved ->
↪  received
  // ...
  @param token The token address to swap from (must be WBTC). // from -> to
```

Recommendation: While these issues do not directly impact the functionality or security of the contract, addressing them can improve the overall readability and consistency of the codebase.

Bitcorn: Corrected most typos in commit 29b51536 as suggested.

Spearbit: Fix verified.