BlendMoney Contracts

Security Review

Cantina Managed review by:

Xmxanuel, Lead Security Researcher
Sujith Somraaj, Security Researcher

August 10, 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

Blend's Intent Engine delivers sustainable yield and auto-derisks your capital.

From Jul 23rd to Jul 28th the Cantina team conducted a review of contracts on commit hash d56e261e. The team identified a total of **25** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	4	4	0
Low Risk	5	4	1
Gas Optimizations	2	2	0
Informational	14	12	2
Total	**25**	**22**	**3**

3 Findings

3.1 Medium Risk

3.1.1 Improper slippage handling by MorphoVaultController during rebalancing

Severity: Medium Risk

Context: MorphoVaultController.sol#L109, MorphoVaultController.sol#L141, MorphoVaultController.sol#L210, MorphoVaultController.sol#L221, MorphoVaultController.sol#L229, MorphoVaultController.sol#L293, MorphoVaultController.sol#L307

Description: The functions `erc4626Deposit`, `morphoRepay`, `morphoBorrow`, `erc4626Redeem`, and `erc4626Withdraw` are invoked throughout the `MorphoVaultController` contract (e.g., during vault rebalancing, collateral adjustments, and flashloan bundles). These calls pass hardcoded values for slippage boundaries via the `maxSharePriceE27` and `minSharePriceE27` parameters:

```
calls[1] = erc4626Deposit(address(GENERAL_ADAPTER), vault, balanceOfSafe, type(uint256).max, address(this));
// ...
calls[0] = erc4626Withdraw(address(GENERAL_ADAPTER), vault, amount, 0, address(GENERAL_ADAPTER), safeAddress);
// ...
callBundle[callIndex++] = morphoRepay(
  address(GENERAL_ADAPTER), marketParams, 0, type(uint256).max, type(uint256).max, safeAddress, hex""
);
// ...
callBundle[callIndex++] = morphoBorrow(
  address(GENERAL_ADAPTER), marketParams, borrowAmountRemaining, 0, 0, address(GENERAL_ADAPTER)
);
// ...
callBundle[callIndex++] = erc4626Redeem(
  address(GENERAL_ADAPTER),
  marketParams.collateralToken,
  type(uint256).max,
  0,
  address(marketConfig.adapter),
  address(GENERAL_ADAPTER)
);
```

These parameters are meant to enforce bounds on the acceptable exchange rate between assets and vault shares during deposits/withdrawals. However:

- `MaxSharePriceE27` is set to `type(uint256).max`, effectively disabling upper bound checks during deposits.
- `MinSharePriceE27` is set to 0, disabling lower bound checks during withdrawals.

This removes all slippage protection and can result in:

- Users receive fewer shares than expected during deposits if the share price is inflated.
- Users receive fewer assets than expected during withdrawals if the vault share price has degraded.

This, under some conditions, could lead to sub-optimal output for the user (or) even loss of funds depending on the type of vault and other parameters.

Recommendation: The existing design already includes a flexible bytes `extraData` field in `RebalanceData`. This can be leveraged to dynamically pass acceptable slippage bounds (`maxSharePriceE27`, `minSharePriceE27`) from the executor to the `MorphoVaultController`.

This removes the need for hardcoded values like type(uint256).max or 0.

BlendMoney: Fixed in PR 42.

Cantina Managed: Fix verified.

3.1.2 `MorphoVaultController.executeRebalance`: insufficient `rebalanceData` validation may cause incorrect leveraged positions

Severity: Medium Risk

Context: StrategyManager.sol#L191

Description: The `executeRebalance` function doesn't sufficiently validate the relationship between `VaultConfig.markets` and `RebalanceData`. The `VaultConfig` can be changed by governance, such as by adding or deleting markets via the `RoleBroadcaster`, while the `executor` continuously calls different `Safes` to trigger `executeRebalance` at certain time intervals. Only comparing the different lengths of the stored `markets` from the `VaultConfig` and the `rebalanceData` is not sufficient:

```
modifier _validateRebalanceData(VaultConfig memory vaultConfig, RebalanceData[] calldata rebalanceData) {
    // Ensure rebalance data matches number of markets in vault config
    require(rebalanceData.length == vaultConfig.markets.length, InvalidRebalanceData());

    _;
}
```

There's no validation that:

1. The `RebalanceData` array corresponds to the correct markets in `VaultConfig`.
2. The order of operations matches the intended markets.
3. The `extraData` is appropriate for the target market.

This allows potential mismatches where rebalance operations could be applied to wrong markets, especially if the market configuration changes between when the `executor` signs the intent and when it's executed.

Recommendation: Add explicit market validation by either:

1. Include `marketId` in `RebalanceData`: Add a `marketId` field to the `RebalanceData` struct to explicitly link each operation to its target market.
2. Version the `VaultConfig`: Add versioning to ensure the configuration hasn't changed since the `executors` intent was signed.

This ensures rebalance operations are always applied to the intended markets and prevents configuration mismatches. Adding `marketId` to `RebalanceData` seems better because it explicitly links each operation to its target market.

BlendMoney: Fixed in PR 37.

Cantina Managed: Fix verified.

3.1.3 A single `ExactErc4626Adapter` griefing attack can continuously block all `executeRebalance` transactions using the adapter

Severity: Medium Risk

Context: ExactErc4626Adapter.sol#L66

Description: The `ExactErc4626Adapter` has a griefing attack vulnerability in its `swapToCollateral` and `swapToLoanToken` functions. An attacker can depositing 1 wei of ERC4626 shares (collateralTokens) and 1 wei of ERC4626 assets (`loanTokens`) directly into the adapter contract, causing all `swapToCollateral` and `swapToLoanToken` to revert each time.

An attacker could iterate the `VaultConfigs` and deposit the 2 wei for each market which uses the `ExactErc4626Adapter` as `IMarketAdapterController`. This would make the `executeRebalance` function not useable anymore and each call would result in a revert, if the market uses the `ExactErc4626Adapter`. The governance would have to trigger a `VaultConfigs` update to replace the `IMarketAdapterController` for the markets. Because governance will be subject to a timelock during this period, the funds are at risk since the executor cannot rebalance the leveraged positions.

The issue occurs for the following reason for the `swapToCollateral` function: if 1 wei share (collateralToken) has been deposited by an attacker:

1. The `_emptyAdapterBalances` modifier requires zero balance of both loan and collateral tokens after execution.
2. The `ERC4626.deposit(amount, recipient)` call transfer shares to the `recipient`, not `address(this)` therefore is no need to transfer the collateralToken like in the `SwapAdapter`.
3. Pre-existing shares from the attacker remain in the contract, failing the empty balance check.

```
modifier _emptyAdapterBalances(IERC20 loanToken, IERC20 collateralToken) {
    _;
    require(loanToken.balanceOf(address(this)) == 0, ZeroAmount());
    require(collateralToken.balanceOf(address(this)) == 0, ZeroAmount()); // Fails with attacker's 1 wei
}
```

The same is true, vice versa for the `swapToLoanToken` function.

- Terminology in the `ExactErc4626Adapter`:

```
collateralTokens => erc4626 shares
loanTokens       => erc4626 asset
deposit          => takes loanTokens (asset) and returns collateralToken(shares) to the receiver
redeem           => takes collateralTokens (shares) and returns loanTokens (assets) to the receiver
```

Recommendation: Since the contract only uses either the asset or shares balances and the other one will be transferred by the ERC4626 to the receiver. Add only one `require` statement to the post check.

- `swapToCollateral`:

```
require(loanToken.balanceOf(address(this)) == 0, ZeroAmount());
```

- `swapToLoanToken`:

```
require(collateralToken.balanceOf(address(this)) == 0, ZeroAmount());
```

BlendMoney: Fixed in PR 41.

Cantina Managed: Fix verified.

3.1.4 `MorphoVaultController._increaseAllocation` can lead to an arithmetic underflow for rebalance allocations in some cases

Severity: Medium Risk

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

Description: The `_increaseAllocation` function contains an arithmetic underflow vulnerability that causes rebalancing operations to revert when `currentBalanceOnGeneralAdapter` exceeds the `MorphoVaultLib._userAmount(marketConfig.leverage, amount)` for the `erc4626Withdraw` parameter calculation.

```
callBundle[callIndex++] = erc4626Withdraw(
    address(GENERAL_ADAPTER),
    vault,
    MorphoVaultLib._userAmount(marketConfig.leverage, amount) - currentBalanceOnGeneralAdapter, // Underflow!
    0,
    address(GENERAL_ADAPTER),
    safeAddress
);
```

The `erc4626Withdraw` is needed to request additional vault tokens to repay the flash loan. However, before the `_increaseAllocation` all previous markets are decreased. The underflow occurs when decrease operations (executed first) deposit more vault tokens into `GENERAL_ADAPTER` than what individual increase operations needs in addition to cover the flash loan repayment.

Example: Executor has 100% allocation in Market A, wants to move 50% in Market B, 50% in Market C by using `executeRebalance`.

- PreState:
  - Market A.
    * Debt: 8B, collateral: 10A.
    * Leverage: 5.
- `executeRebalance`:
  - `DecreaseAllocation` market A 100%.

    * Fully closing the position would move 2A (2e18) to the generalAdapter.
  - `_increaseAllocation` market B 50%.
    * 1A should be moved into market B with 5x leverage.
    * `_increaseAllocation` amount parameter = 5A.
  - CurrentBalanceOnGeneralAdapter snapshot before bundle call = 2A.
  - `MorphoVaultLib._userAmount(5e18, 5e18) = 1e18.`
  - Underflow would be: `MorphoVaultLib._userAmount(5e18, 5e18) - currentBalanceOnGeneralAdapter = 1e18 - 2e18` ⇒ `underflow.`

Recommendation: Add a check to prevent underflow and skip unnecessary withdrawals:

```
uint256 requiredVaultTokens = MorphoVaultLib._userAmount(marketConfig.leverage, amount);
if (requiredVaultTokens > currentBalanceOnGeneralAdapter) {
    callBundle[callIndex++] = erc4626Withdraw(
        address(GENERAL_ADAPTER),
        vault,
        requiredVaultTokens - currentBalanceOnGeneralAdapter,
        0,
        address(GENERAL_ADAPTER),
        safeAddress
    );
}
```

BlendMoney: Fixed in PR 34.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Override and disable `setPeer` in `RolesReceiver`

Severity: Low Risk

Context: RolesReceiver.sol#L108

Description: The `RolesReceiver` contract can only accept incoming messages from the trusted broadcaster deployed in the trusted chain ID. These configurations are made immutable in the constructor. Since the sender is expected to be a specific address from a particular chain ID, the `setPeer()` functionality is redundant and can introduce DoS if misused by the owner.

Proof of Concept: The following Proof of Concept demonstrates the issue, where the sender is the trusted broadcaster, but due to bad peer configuration, the inbound message is blocked:

```
pragma solidity ^0.8.25;

import "forge-std/Test.sol";

import {OptionsBuilder} from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OptionsBuilder.sol";
import {LayerZeroV2Helper} from "lib/pigeon/src/layerzero-v2/LayerZeroV2Helper.sol";

import "src/RolesBroadcaster.sol";
import "src/RolesReceiver.sol";

contract AuditTest is Test {
    using OptionsBuilder for bytes;
    LayerZeroV2Helper public lzHelper;

    uint256 ethForkId;
    uint256 baseForkId;

    RolesBroadcaster public sender;
    RolesReceiver public receiver;

    address owner = makeAddr("OWNER");
    address executor = makeAddr("EXECUTOR");

    function setUp() external {
        ethForkId = vm.createSelectFork(vm.envString("ETH_RPC_URL"), 23015625);
        deal(owner, 100 ether);
        lzHelper = new LayerZeroV2Helper();
        sender = new RolesBroadcaster(
            0x1a44076050125825900e736c501f859c50fE728c, // endpoint
            owner // delegate
        );

        baseForkId = vm.createSelectFork(vm.envString("BASE_RPC_URL"), 33446000);
        receiver = new RolesReceiver(
            0x1a44076050125825900e736c501f859c50fE728c, // endpoint
            owner,
            address(sender),
            uint32(30101)
        );
        vm.prank(owner);
        receiver.setPeer(uint32(30101), bytes32(uint256(uint160(address(520)))));

        vm.selectFork(ethForkId);
        vm.startPrank(owner);
        sender.setPeer(uint32(30184), bytes32(uint256(uint160(address(receiver)))));
        sender.setRemoteReceiver(uint32(30184), address(receiver));
        vm.stopPrank();
    }

    function test_audit() external {
        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);
        bytes memory callData = abi.encodeWithSignature("setExecutor(address)", executor);

        vm.recordLogs();
        vm.prank(owner);
        sender.send{value: 1 ether}(uint32(30184), callData, options);
        lzHelper.help(0x1a44076050125825900e736c501f859c50fE728c, baseForkId, vm.getRecordedLogs());

        vm.selectFork(baseForkId);
        assert(receiver.executor() == executor);
    }
}
```

Recommendation: Consider disabling the `setPeer()` function as follows:

```
contract RolesReceiver is OApp, StrategyManager {
  // ....
  constructor(address _lzEndpoint, address _owner, address _trustedBroadcaster, uint32 _trustedChainId)
    OApp(_lzEndpoint, _owner)
    Ownable(_owner) {
     // ....
     _setPeer(_trustedChainId, bytes32(uint256(uint160(_trustedBroadcaster))));
   }

   // ....

   function setPeer(uint32 eid, bytes32 peer) public override {
       revert NOT_SUPPORTED();
   }

   // ....
}
```

BlendMoney: Fixed in PR 31.

Cantina Managed: Fix verified.

3.2.2 Malformed message data can result in incorrect decoding and lead to unsafe state changes

Severity: Low Risk

Context: RolesReceiver.sol#L124, RolesReceiver.sol#L129

Description: The `_lzReceive()` function in the `RolesReceiver` contract assumes that incoming message data from LayerZero is correctly formatted and sufficiently long for decoding. However, there are no assertions in the `message.length` before decoding or using calldataload with assembly. This can result in potential unauthorized or unintended behavior, such as setting the executor to an arbitrary or null address (or) setting vault configuration for a bad vault address.

Recommendation: Add explicit length checks before decoding the message, based on the expected format for each supported selector.

BlendMoney: Fixed in PR 23.

Cantina Managed: The issue is partially fixed as length checks are added only to the `_SET_EXECUTOR_SELECTOR` branch. For the `_UPDATE_VAULT_CONFIG_SELECTOR` branch, no checks were enforced due to the dynamic nature of the type. However a base-line sanity check is suggested.

3.2.3 Unnecessary token approvals in `MorphoVaultController._increaseAllocation` increase attack surface

Severity: Low Risk

Context: MorphoVaultController.sol#L238

Description: The `_increaseAllocation` unnecessarily approves both `vaultToken` and `collateralToken` to `GENERAL_ADAPTER`. These approvals are not required and should be removed to reduce complexity and lower the security risk:

```
ERC20(vaultToken).forceApprove(address(GENERAL_ADAPTER), type(uint256).max);
IERC20(marketParams.collateralToken).forceApprove(address(GENERAL_ADAPTER), type(uint256).max);
```

The approval are reset to zero after the `Bundle` call which executes the batch of transactions in the `GENERAL_ADAPTER`. However, an attack vector still exists for example, a malicious (or upgraded) ERC20 `collateral` token implemenation could exploit the `ERC20.approve` process and call `GENERAL_ADAPTER.erc20TransferFrom` to steal all `vaultToken` from a `Safe`.

Note: The `ERC20.approve` calls are not happening as part of a Bundler multicall. This means the `GENERAL_ADAPTER` is not in lock mode and can be called.

Recommendation: Remove the unnecessary approvals and the reset to zero. Only grant approval when it is truly necessary.

BlendMoney: Fixed in PR 30.

Cantina Managed: Fix verified.

3.2.4 Privileged role and actions lead to centralization risks for users

Severity: Low Risk

Context: RolesBroadcaster.sol#L56, RolesReceiver.sol#L83

Description: Several `onlyOwner` functions across the protocol affect critical protocol state and semantics, and therefore, lead to centralization risk for users. Some examples are highlighted below:

- `setPeer()` in `RolesReceiver` can only be called by the owner. Hence, by setting a wrong address, the owner can delay all incoming configuration changes from `RolesBroadcaster`.
- The delegate address in both `RolesReceiver` and `RolesBroadcaster` has significant control over the OApp configuration parameters like DVNs, send and receive libraries, etc..

Recommendation: Consider:

- Documenting the privileged role and actions for protocol user awareness.
- Privilege actions affecting critical protocol semantics should be locked behind timelocks so that users can decide to exit or engage.
- Following the strictest opsec guidelines for privileged keys, e.g., use of reasonable multi-signature and hardware wallets.

Blend Money:

- `setPeer()` in `RolesReceiver` is removed in PR 39.
- The delegate address will serve as a time lock to prevent unforeseen configuration modifications and to offer users a transparent, pre-established timeline for making decisions.

Cantina Managed: Acknowledged.

3.2.5 No parameter sanity checks in `StrategyManager._updateStrategy` for duplicates markets or correct `leverage` values

Severity: Low Risk

Context: StrategyManager.sol#L163

Description: The `StrategyManager._updateStrategy` allows to update the markets used for the leverage positions for a specific vault. However, there a no sanity checks as a last defense mechanism to prevent a incorrect governance `updateVaultConfig` call.

Recommendation: Require the `markets` array to be sorted by `marketId` to easily identify duplicates. Check if `market.leverage` is > `1e18` and smaller than a reasonable upper limit.

BlendMoney: Fixed in PR 36.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 `TRUSTED_BROADCASTER` can be `bytes32`

Severity: Gas Optimization

Context: RolesReceiver.sol#L38, RolesReceiver.sol#L118

Description: An immutable variable `TRUSTED_BROADCASTER` is declared in `RolesReceiver` as type `address`. However, it could be declared `bytes32` to avoid casting every time for validation in the `_lzReceive()` function.

Recommendation: Consider declaring `TRUSTED_BROADCASTER` as `bytes32`.

BlendMoney: Fixed in PR 31.

Cantina Managed: Fix verified.

3.3.2 `RolesReceiver` in `MarketWrapper` could be made immutable

Severity: Gas Optimization

Context: MarketWrapper.sol#L25

Description: The state variable `rolesReceiver` is only set in the constructor and remains unchanged later. Such variables can be declared immutable for efficiency and clarity.

Recommendation: Consider declaring the variable `rolesReceiver` as immutable.

BlendMoney: Fixed in PR 21.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Enforce message ordering in `RolesReceiver`

Severity: Informational

Context: RolesReceiver.sol#L18

Description: Blend utilizes LayerZero to configure contracts cross-chain from a single source chain, referred to as a trusted chain ID. The `RolesBroadcaster` contract is deployed to the trusted chain ID, which can send messages to `RolesReceiver` contracts deployed across multiple chains that Blend supports.

By default, LayerZero's OApp library enforces no message ordering, which can lead to race conditions for Blend. For example, the timelock (or) governance initiates two messages relatively close together during the same time; then, by re-ordering them, configurations could end up differently than expected by the protocol.

Proof of Concept: Created a Proof of Concept by sending two messages to set the executor addresses at the same time. By re-ordering the configurations, they end up different from what was expected. The expected output should be that the executor address should be `executor2` instead. Due to reordering, we end up with an outdated executor address.

```
pragma solidity ^0.8.25;

import "forge-std/Test.sol";

import {OptionsBuilder} from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OptionsBuilder.sol";
import {LayerZeroV2Helper} from "lib/pigeon/src/layerzero-v2/LayerZeroV2Helper.sol";

import "src/RolesBroadcaster.sol";
import "src/RolesReceiver.sol";

contract AuditTest is Test {
    using OptionsBuilder for bytes;
    LayerZeroV2Helper public lzHelper;

    uint256 ethForkId;
    uint256 baseForkId;

    RolesBroadcaster public sender;
    RolesReceiver public receiver;

    address owner = makeAddr("OWNER");
    address executor1 = makeAddr("EXECUTOR_1");
    address executor2 = makeAddr("EXECUTOR_2");

    function setUp() external {
        ethForkId = vm.createSelectFork(vm.envString("ETH_RPC_URL"), 23015625);
        deal(owner, 100 ether);
        lzHelper = new LayerZeroV2Helper();
        sender = new RolesBroadcaster(
            0x1a44076050125825900e736c501f859c50fE728c, // endpoint
            owner // delegate
        );

        baseForkId = vm.createSelectFork(vm.envString("BASE_RPC_URL"), 33446000);
        receiver = new RolesReceiver(
            0x1a44076050125825900e736c501f859c50fE728c, // endpoint
            owner,
            address(sender),
            uint32(30101)
        );

        vm.prank(owner);
        receiver.setPeer(30101, bytes32(uint256(uint160(address(sender)))));

        vm.selectFork(ethForkId);
        vm.startPrank(owner);
        sender.setPeer(uint32(30184), bytes32(uint256(uint160(address(receiver)))));
        sender.setRemoteReceiver(uint32(30184), address(receiver));
        vm.stopPrank();
    }

    function test_audit() external {
        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);


        bytes memory callData1 = abi.encodeWithSignature("setExecutor(address)", executor1);
        bytes memory callData2 = abi.encodeWithSignature("setExecutor(address)", executor2);

        vm.recordLogs();
        vm.prank(owner);
        sender.send{value: 1 ether}(uint32(30184), callData1, options);
        Vm.Log[] memory logs1 = vm.getRecordedLogs();

        vm.recordLogs();
        vm.prank(owner);
        sender.send{value: 1 ether}(uint32(30184), callData2, options);
        Vm.Log[] memory logs2 = vm.getRecordedLogs();

        lzHelper.help(0x1a44076050125825900e736c501f859c50fE728c, baseForkId, logs2);
        lzHelper.help(0x1a44076050125825900e736c501f859c50fE728c, baseForkId, logs1);

        vm.selectFork(baseForkId);
        assert(receiver.executor() == executor1);
    }
}
```

Recommendation: Consider enforcing ordered delivery to ensure messages aren't re-ordered by relayer. For more information, refer to LayerZero documentation here

Blend Money: We strongly believe this issue is informational, given the broadcaster's infrequent transmission schedule (approximately 1/2 per week). Given the above, we prefer to err on the side of avoiding a governance sync + the added nonce complexity.

Cantina: The issue was downgraded from low severity to informational because the messaging process is strictly access-controlled, includes a time lock, and has been acknowledged.

3.4.2 Remove `remoteReceiverAddresses` mapping from `RolesBroadcaster`

Severity: Informational

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

Description: The `remoteReceiverAddresses` mapping in `RolesBroadcaster` holds the same data as the `peers` mapping in the underlying OApp contract but uses a different data type.

Recommendation: Since its redundant, would suggest removing the `remoteReceiverAddresses` mapping altogether:

```diff
  contract RolesBroadcaster is OApp, OAppOptionsType3 {
-     mapping(uint32 => address) public remoteReceiverAddresses;
      // ....
-     error ReceiverAddressNotSet();
       // ....
-     function setRemoteReceiver(uint32 chainId, address receiverAddress) external onlyOwner {
-         remoteReceiverAddresses[chainId] = receiverAddress;
-         emit RemoteReceiverSet(chainId, receiverAddress);
-     }
       // ....
-     function send(uint32 _dstEid, bytes calldata _callData, bytes calldata _options) external payable
↪   onlyOwner {
-       address receiverAddress = remoteReceiverAddresses[_dstEid];
-       require(receiverAddress != address(0), ReceiverAddressNotSet());
          // ....
-     }
  }
```

Blend Money: Fixed in PR-17 and PR-44

BlendMoney: Fixed in PR 17 and PR 44.

Cantina Managed: Fix verified.

3.4.3 Remove unused file imports

Severity: Informational

Context: CallBuilder.sol#L8-L9, MorphoVaultLib.sol#L4

Description:

- Two file imports, `ERC20WrapperAdapter` and `IERC20`, remain unused in the `CallBuilder.sol`, which could be removed.
- `MathRayLib` is imported by `MorphoVaultLib` but is redundant and could be removed as well.

Recommendation: Consider removing the unused file import.

BlendMoney: Fixed in PR 29.

Cantina Managed: Fix verified.

3.4.4 `RolesGuard` missing newer Safe `checkModuleTransaction` interface

Severity: Informational

Context: RolesGuard.sol#L79

Description: The RolesGuard only implements the legacy Safe guard interface. Newer Safe versions expect a `checkModuleTransaction` function for module transactions, which has a different signature (see ModuleManager.sol#L15). Without this interface, the `Guard` can't be set to a `ModuleGuard` in newer Safes version.

Recommendation: Consider this change if `blend.money` wants to support newer safe versions module-specific guard interface:

```
function checkModuleTransaction(
    address to,
    uint256 value,
    bytes memory data,
    Enum.Operation operation,
    address module
) external returns (bytes32 moduleTxHash) {
    require(!paused(), SystemPaused());
    return bytes32(0);
}
```

Note: The `ModuleGuard` is only used for the module and not for regular Safe transactions. Therefore, only a `!paused()` check is required.

BlendMoney: Fixed in PR 38.

Cantina Managed: Fix verified.

3.4.5 Inconsistent naming of `_UPDATE_STRATEGY_CONFIG_SELECTOR` in `RolesReceiver._updateStrategy`

Severity: Informational

Context: RolesReceiver.sol#L122

Description: The internal function `_updateStrategy` is called when the message equals the `_UPDATE_STRATEGY_CONFIG_SELECTOR` constant, which has the value:

```
bytes4 private constant _UPDATE_STRATEGY_CONFIG_SELECTOR =
    bytes4(keccak256("updateVaultConfig(address,(address,(bytes32,address,uint256,bool,bytes)[],(address)[]))"
    ↪  ));
```

This is a naming inconsistency between constant function name `updateVaultConfig` and the actual function called `_updateStrategy`.

Recommendation: Rename the internal function to match the constants name or vice-versa.

```
// Change from:
_updateStrategy(vaultAddress, configs);

// To:
_updateVaultConfig(vaultAddress, configs);
```

This improves code readability and maintains consistency.

BlendMoney: Fixed in PR 25.

Cantina Managed: Fix verified.

3.4.6 `MorphoVaultController` functions can be called without a `delegate_call` creating a risk of fund loss

Severity: Informational

Context: MorphoVaultController.sol#L84

Description: The `MorphoVaultController` is intended to be called only via `delegate_call` from a `Safe`. However, a user could still transfer funds directly to the `MorphoVaultController` and invoke its functions using a regular `CALL`. This creates a security risk: for example, `vaultShares` could remain trapped in the `MorphoVaultController`, and leveraged positions would be owned by the contract itself rather than the intended owner. As a result, anyone could subsequently call the `MorphoVaultController` again and attempt to access those funds.

Recommendation: Because calling the `MorphoVaultController` with a regular `CALL` is not intended and has a security risk, restrict the contract so it can only be called via `delegate_call`. This can be achieved with the following pattern:

```
The `modifier onlyDelegateCall() {
  if (address(this) == SELF) revert OnlyDelegateCall();
  _;
}

address internal immutable SELF;

constructor() {
  SELF = address(this);
}
```

BlendMoney: Fixed in PR 18.

Cantina Managed: Fix verified.

3.4.7 Incorrect interface usage between `IERC4626` and `IERC20` in `MorphoVaultController`

Severity: Informational

Context: MorphoVaultController.sol#L105

Description: The contract incorrectly casts `vaultToken` to `IERC4626` when checking balances in two different places.

```
uint256 balanceOfSafe = IERC4626(vaultToken).balanceOf(address(this));
// ...
uint256 balanceOfGeneralAdapter = IERC4626(vaultToken).balanceOf(address(GENERAL_ADAPTER));
```

The `vaultToken` is the underlying asset ERC20 token (obtained via `IERC4626(vault).asset()`), not the vault itself. While the `asset` could theoretically be another ERC4626, it's cleaner and more consistent to only expect the `IERC20` interface.

Recommendation: Use the IERC20 interface consistently:

```
- uint256 balanceOfSafe = IERC4626(vaultToken).balanceOf(address(this));
+ uint256 balanceOfSafe = IERC20(vaultToken).balanceOf(address(this));
  /...
- uint256 balanceOfGeneralAdapter = IERC4626(vaultToken).balanceOf(address(GENERAL_ADAPTER));
+ uint256 balanceOfGeneralAdapter = IERC20(vaultToken).balanceOf(address(GENERAL_ADAPTER));
```

BlendMoney: Fixed in PR 19.

Cantina Managed: Fix verified.

3.4.8 `MorphoVaultController._increaseAllocation` incorrect comment for `isWrapped` case

Severity: Informational

Context: MorphoVaultController.sol#L208

Description: The comment at line 208 is incorrect:

```
// If it's a wrapper, approve underlying tokens and deposit to get wrapper tokens
if (isWrapped) {
    callBundle[callIndex++] = erc4626Deposit(
        address(GENERAL_ADAPTER),
        marketParams.collateralToken,
        type(uint256).max,
        type(uint256).max,
        address(GENERAL_ADAPTER)
    );
}
```

No approval of underlying tokens is happening or needed. The GeneralAdapter already holds the underlying tokens (e.g., Dai) after the `swapToCollateral` operation, so it can directly deposit them to get wrapper tokens (e.g., sDai).

Recommendation: Update the comment to accurately reflect the operation:

```
// If it's a wrapper, deposit underlying tokens to get wrapper tokens
```

BlendMoney: Fixed in PR 24.

Cantina Managed: Fix verified.

3.4.9 `MorphoVaultController._decreaseAllocation` redundant validation for the `isPositionClosure = true` case

Severity: Informational

Context: MorphoVaultController.sol#L329

Description: The validation at line 329 is redundant:

```
// Sanity check: amount must be less than total supplied collateral
require(amount < totalSuppliedCollateral, InvalidAmount());
```

This same validation already occurs earlier in the function.

```
// Validate withdrawal amount
bool isPositionClosure = amount == type(uint256).max;
require(isPositionClosure || amount < totalSuppliedCollateral, InvalidAmount());
```

Recommendation: Remove the redundant validation to reduce gas costs and code complexity:

```
// Remove the redundant check at line 329
// require(amount < totalSuppliedCollateral, InvalidAmount());
```

BlendMoney: Fixed in PR 20.

Cantina Managed: Fix verified.

3.4.10 Use `Ownable2Step` instead of `Ownable`

Severity: Informational

Context: AssignmentManager.sol#L5, RolesBroadcaster.sol#L6, RolesReceiver.sol#L7

Description: The contracts `AssignmentManager`, `RolesReceiver`, and `RolesBroadcaster` utilize the `Ownable` library from OpenZeppelin, which lacks safeguards during ownership transfers. This approach is considered less secure compared to OpenZeppelin's `Ownable2Step` implementation.

Recommendation: Consider using Ownable2Step instead of Ownable.

Blend Money: We're not able to do this because we need `Ownable` for `OApp` and replacing `Ownable` in the constructor call of the `RolesBroadcaster` and `RolesReceiver` leads to a can of errors. We will remove `AssignmentManager` entirely.

Cantina Managed: Acknowledged.

3.4.11 Redundant roles array in `AssignmentManager`

Severity: Informational

Context: AssignmentManager.sol#L20-L22, AssignmentManager.sol#L83, AssignmentManager.sol#L111

Description: The `AssignmentManager` contract uses a roles array to maintain a list of assignable role identifiers and restricts the use of `setRole()` to only those roles included in this array.

This contract is exclusively used by `RolesGuard`, with only one role, `PAUSER_ROLE`, utilized throughout the entire system.

This introduces unnecessary complexity and gas overhead:

- Every call to `setRole()` involves iterating over the roles array.
- The constructor copies all role identifiers to storage, even though only one role is used.
- The roleExists modifier adds runtime cost with minimal value in a single-role context.

Since there is only one meaningful role in use, the dynamic roles array and the associated roleExists logic serve no real purpose and could be removed in favor of a direct check against a constant. If you need to maintain multiple pausers, could store a mapping of addresses with roles, to reduce complexity and improve efficiency.

Recommendation: Consider refactoring the Assignment manager to allow owner to set new pausers (or) just use the access control library, which is more efficient.

Blend Money: Fixed in PR 38 and PR 45.

Cantina Managed: Fix verified.

3.4.12 Redundant inline documentation

Severity: Informational

Context: AssignmentManager.sol#L50, RolesGuard.sol#L52-L53, RolesGuard.sol#L61-L62

Description:

- The `AssignmentManager` constructor does not set the `DEFAULT_ADMIN_ROLE` in its constructor, despite inline documentation suggesting it does.
- The `pause()` and `unpause()` functions in `RolesGuard` does not affect the `_lzReceive` functionality, but only block the safe transactions.

Recommendation: Consider removing/fixing the above-mentioned inaccurate documentation.

BlendMoney: Fixed in PR 32.

Cantina Managed: Fix verified.

3.4.13 Remove redundant `_isErc20Wrapper` function

Severity: Informational

Context: MorphoVaultController.sol#L367-L374

Description: The internal function `_isErc20Wrapper()` is redundant and remains unused across the entire codebase.

Recommendation: Consider removing the above-mentioned unused function.

BlendMoney: Fixed in PR 28.

Cantina Managed: Fix verified.

3.4.14 Redundant boolean returned in allocation functions

Severity: Informational

Context: MorphoVaultController.sol#L248, MorphoVaultController.sol#L346

Description: The internal allocation helpers `_decreaseAllocation()` and `_increaseAllocation()` are used to process rebalancing inside the `MorphoVaultController` contract. These two functions return a boolean indicating successful execution; however, this value is never utilized within the protocol and is effectively discarded.

Recommendation: Consider removing the redundant boolean returned by the two functions mentioned above.

BlendMoney: Fixed in PR 33.

Cantina Managed: Fix verified.