

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2026.06.23, the SlowMist security team received the PrimeUSD team's security audit application for Boring Vault, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit

Serial Number	Audit Class	Audit Subclass
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This is the Boring Vault contract of the PrimeUSD protocol. The audit mainly includes the Atomic Queue, Vault, Roles, DecodersAndSanitizers, Oracle, and Timelock modules.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Comment Does Not Match Actual Logic	Others	Suggestion	Fixed
N2	`safeUpdateAtomicRequest` Does Not Validate an Accountant Allowlist	Design Logic Audit	Suggestion	Acknowledged
N3	Missing Zero-Address Validation	Others	Suggestion	Fixed
N4	Incorrect ZkLighter Public-Key Length Check Makes the Emergency Key-Rotation Path Unusable	Design Logic Audit	High	Fixed
N5	Misspelled `flashLoan` Function Signature in the Merkle Leaf Generator Invalidates Proofs	Design Logic Audit	Low	Fixed
N6	Missing Events for Important Parameter Updates	Others	Suggestion	Fixed
N7	Redundant Code	Others	Suggestion	Fixed
N8	`cancelTransaction` Does Not Check	Design Logic Audit	Low	Fixed

NO	Title	Category	Level	Status
	Whether the Transaction Is Queued			
N9	`ManagerWithMerkleVerification` Flash Loan Callback Reuses a Shared Root and May Bypass Strategist Permission Isolation	Design Logic Audit	Medium	Fixed
N10	Timelock Canceller Role Can Permanently Veto Governance and Cannot Be Forcibly Removed	Authority Control Vulnerability Audit	Medium	Fixed
N11	Cross-Chain Deposit Flow Depends on a Zero Share-Lock Period Configuration	Design Logic Audit	Information	Acknowledged
N12	Missing Decimal Compatibility Check	Design Logic Audit	Suggestion	Fixed
N13	`AtomicQueue` Solvability Checks Ignore Locked Vault Shares	Design Logic Audit	Information	Acknowledged
N14	Risk of Excessive Privilege	Authority Control Vulnerability Audit	Medium	Acknowledged
N15	`MultiChainTellerBase` Missing `minimumMessageGas <= messageGasLimit` Validation	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/SwellNetwork/boring-vault

Initial Audit Version: 7afe61e23e669710b7e3dc0818856d36ce9b7c3a

Final Audit Version: 54cf6354c479dc2e54cc7a29424d4afaac50eda2

Audit Scope:

```
./src
├── atomic-queue
│   ├── AtomicQueue.sol
│   └── AtomicSolverV3.sol
├── base
│   ├── BoringVault.sol
│   ├── DecodersAndSanitizers
│   │   ├── BaseDecoderAndSanitizer.sol
│   │   ├── Protocols
│   │   │   └── AaveV3DecoderAndSanitizer.sol
│   │   └── hyperwave
│   │       ├── aprimeusd
│   │       │   ├── EmergencyZkLighterDecoderAndSanitizer.sol
│   │       │   └── ZkLighterDecoderAndSanitizer.sol
│   │       └── common
│   │           ├── AaveV3DecoderAndSanitizer.sol
│   │           ├── AccountantDecoderAndSanitizer.sol
│   │           └── KyberswapDecoderAndSanitizer.sol
│   ├── Roles
│   │   ├── AccountantWithRateProviders.sol
│   │   ├── CrossChain
│   │   │   ├── CrossChainTellerBase.sol
│   │   │   ├── MultiChainLayerZeroTellerWithMultiAssetSupport.sol
│   │   │   └── MultiChainTellerBase.sol
│   │   ├── CrossChainAdapter.sol
│   │   ├── ManagerWithMerkleVerification.sol
│   │   └── TellerWithMultiAssetSupport.sol
│   └── aprimeUSD
│       ├── Configuration.sol
│       └── EmergencyExecutor.sol
├── oracles
│   └── RedstoneStablecoinRateProvider.sol
└── timelock
    └── Timelock.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

AtomicQueue			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Auth
pause	External	Can Modify State	requiresAuth
unpause	External	Can Modify State	requiresAuth
getUserAtomicRequest	External	-	-
isAtomicRequestValid	External	-	-
updateAtomicRequest	External	Can Modify State	nonReentrant requiresAuth
safeUpdateAtomicRequest	External	Can Modify State	nonReentrant requiresAuth
solve	External	Can Modify State	nonReentrant requiresAuth
viewSolveMetaData	External	-	-
viewVerboseSolveMetaData	External	-	-
viewRequestTiming	External	-	-
setRequestUpdateCooldown	External	Can Modify State	requiresAuth
setMinimumRequestAge	External	Can Modify State	requiresAuth
_updateAtomicRequest	Internal	Can Modify State	-

AtomicQueue			
_calculateAssetAmount	Internal	-	-

AtomicSolverV3			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Auth
rescueTokens	External	Can Modify State	requiresAuth
p2pSolve	External	Can Modify State	requiresAuth
redeemSolve	External	Can Modify State	requiresAuth
finishSolve	External	Can Modify State	requiresAuth
_p2pSolve	Internal	Can Modify State	-
_redeemSolve	Internal	Can Modify State	-

BoringVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 Auth
manage	External	Can Modify State	requiresAuth
manage	External	Can Modify State	requiresAuth
enter	External	Can Modify State	requiresAuth
exit	External	Can Modify State	requiresAuth
setBeforeTransferHook	External	Can Modify State	requiresAuth
_callBeforeTransfer	Internal	-	-

BoringVault			
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
<Receive Ether>	External	Payable	-

BaseDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
approve	External	-	-
acceptOwnership	External	-	-
transferOwnership	External	-	-
<Fallback>	External	Can Modify State	-

AaveV3DecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
supply	External	-	-
withdraw	External	-	-
borrow	External	-	-
repay	External	-	-
setUserUseReserveAsCollateral	External	-	-
setUserEMode	External	-	-

EmergencyZkLighterDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseDecoderAndSanitizer
burnShares	External	-	-
changePubKey	External	-	-

ZkLighterDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseDecoderAndSanitizer
deposit	External	-	-

AaveV3ImplDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseDecoderAndSanitizer

AccountantDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseDecoderAndSanitizer
claimFees	External	-	-

KyberswapDecoderAndSanitizer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseDecoderAndSanitizer(_boringVault)

KyberswapDecoderAndSanitizer			
swap	External	Payable / Can Modify State	-
setMaxSlippage	External	Can Modify State	onlyRole(GUARDIAN_ROLE)
_validateMaxSlippageSetting	Internal	-	-
_validateSlippage	Internal	-	-

AccountantWithRateProviders			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Auth
pause	External	Can Modify State	requiresAuth
unpause	External	Can Modify State	requiresAuth
updateDelay	External	Can Modify State	requiresAuth
updateUpper	External	Can Modify State	requiresAuth
updateLower	External	Can Modify State	requiresAuth
updateManagementFee	External	Can Modify State	requiresAuth
updatePayoutAddress	External	Can Modify State	requiresAuth
setRateProviderData	External	Can Modify State	requiresAuth
updateExchangeRate	External	Can Modify State	requiresAuth
claimFees	External	Can Modify State	-
getRate	Public	-	-
getRateSafe	External	-	-

AccountantWithRateProviders			
getRateInQuote	Public	-	-
getRateInQuoteSafe	External	-	-
changeDecimals	Internal	-	-

CrossChainTellerBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
depositAndBridge	External	Payable / Can Modify State	requiresAuth nonReentrant
previewFee	External	-	-
bridge	Public	Payable / Can Modify State	requiresAuth
_bridge	Internal	Can Modify State	-
_quote	Internal	-	-
_beforeBridge	Internal	Can Modify State	-
_afterBridge	Internal	Can Modify State	-
_beforeReceive	Internal	Can Modify State	-
_afterReceive	Internal	Can Modify State	-

MultiChainLayerZeroTellerWithMultiAssetSupport			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_quote	Internal	-	-

MultiChainLayerZeroTellerWithMultiAssetSupport			
_lzReceive	Internal	Can Modify State	-
_bridge	Internal	Can Modify State	-

MultiChainTellerBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addChain	External	Can Modify State	requiresAuth
stopMessagesFromChain	External	Can Modify State	requiresAuth
allowMessagesFromChain	External	Can Modify State	requiresAuth
removeChain	External	Can Modify State	requiresAuth
allowMessagesToChain	External	Can Modify State	requiresAuth
stopMessagesToChain	External	Can Modify State	requiresAuth
setChainGasLimit	External	Can Modify State	requiresAuth
_beforeBridge	Internal	Can Modify State	-

CrossChainAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_deposit	Internal	Can Modify State	-
lzCompose	External	Payable / Can Modify State	nonReentrant
deposit	External	Can Modify State	requiresAuth nonReentrant

CrossChainAdapter			
updateAtomicRequest	External	Can Modify State	requiresAuth nonReentrant
bridge	External	Can Modify State	requiresAuth nonReentrant
rescueERC20	External	Can Modify State	requiresAuth
rescueNative	External	Can Modify State	requiresAuth
<receive>	External	Payable / Can Modify State	-

ManagerWithMerkleVerification			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setManageRoot	External	Can Modify State	requiresAuth
pause	External	Can Modify State	requiresAuth
unpause	External	Can Modify State	requiresAuth
manageVaultWithMerkleVerification	External	Can Modify State	requiresAuth
flashLoan	External	Can Modify State	-
receiveFlashLoan	External	Can Modify State	-
_verifyCallData	Internal	-	-
_verifyManageProof	Internal	-	-

TellerWithMultiAssetSupport			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

TellerWithMultiAssetSupport			
pause	External	Can Modify State	requiresAuth
unpause	External	Can Modify State	requiresAuth
addAsset	External	Can Modify State	requiresAuth
removeAsset	External	Can Modify State	requiresAuth
setShareLockPeriod	External	Can Modify State	requiresAuth
beforeTransfer	Public	-	-
refundDeposit	External	Can Modify State	requiresAuth
deposit	External	Can Modify State	requiresAuth nonReentrant
depositWithPermit	External	Can Modify State	requiresAuth nonReentrant
bulkDeposit	External	Can Modify State	requiresAuth nonReentrant
bulkWithdraw	External	Can Modify State	requiresAuth
_erc20Deposit	Internal	Can Modify State	-
_afterPublicDeposit	Internal	Can Modify State	-

Configuration			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setConfiguration	External	Can Modify State	requiresAuth
getConfiguration	External	-	-

EmergencyExecutor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
aaveWithdraw	External	Can Modify State	requiresAuth nonReentrant
zkLighterBurnShares	External	Can Modify State	requiresAuth nonReentrant
zkLighterChangePubKey	External	Can Modify State	requiresAuth nonReentrant
_executeSingleCall	Internal	Can Modify State	-
_getProofUsingTree	Internal	-	-
_generateProof	Internal	-	-
_hashPair	Private	-	-
_efficientHash	Private	-	-

RedstoneStablecoinRateProvider			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setLowerBound	External	Can Modify State	onlyRole(GOVERNANCE_ROLE)
setMaxRate	External	Can Modify State	onlyRole(GOVERNANCE_ROLE)
setSequencerDowntimeLimit	External	Can Modify State	onlyRole(GOVERNANCE_ROLE)
updateSequencerStatus	External	Can Modify State	onlyRole(GUARDIAN_ROLE)
_calculateScale	Internal	-	-
_calculateRate	Internal	-	-
getRate	Public	-	-

RedstoneStablecoinRateProvider			
_rateCheck	Internal	-	-
_isEqual	Internal	-	-
_isEqual	Internal	-	-
_checkSequencerDowntime	Internal	-	-

Timelock			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<fallback>	External	Payable / Can Modify State	-
setDelay	Public	Can Modify State	-
acceptAdmin	Public	Can Modify State	-
setPendingAdmin	Public	Can Modify State	-
setCanceller	Public	Can Modify State	-
queueTransaction	Public	Can Modify State	-
cancelTransaction	Public	Can Modify State	-
executeTransaction	Public	Payable / Can Modify State	-
getBlockTimestamp	Internal	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Comment Does Not Match Actual Logic

Category: Others

Content

In `AtomicQueue`, `isPaused` actually blocks the two core operations of user request updates and solver matching.

However, the comments above `pause` / `unpause` refer to `updateExchangeRate`, which does not exist in this contract.

- src/atomic-queue/AtomicQueue.sol#L206-L223

```
/**
 * @notice Pause this contract, which prevents future calls to `updateExchangeRate`,
and any safe rate
 *         calls will revert.
 * @dev Callable by MULTISIG_ROLE.
 */
function pause() external requiresAuth {
    isPaused = true;
    emit Paused();
}

/**
 * @notice Unpause this contract, which allows future calls to `updateExchangeRate`,
and any safe rate
 *         calls will stop reverting.
 * @dev Callable by MULTISIG_ROLE.
 */
function unpause() external requiresAuth {
    isPaused = false;
    emit Unpaused();
}
```

Solution

Update the comments so they accurately describe how the paused state affects request updates and solver matching in `AtomicQueue`.

Status

Fixed

[N2] [Suggestion] `safeUpdateAtomicRequest` Does Not Validate an Accountant Allowlist

Category: Design Logic Audit

Content

In `AtomicQueue`, `safeUpdateAtomicRequest` does not validate `accountant` against an allowlist. If a user mistakenly provides a malicious `AccountantWithRateProviders` contract, the user may unknowingly sell the offer token at an unfavorable price.

- src/atomic-queue/AtomicQueue.sol#L300-L330

```
function safeUpdateAtomicRequest(
    ERC20 offer,
    ERC20 want,
    AtomicRequest memory userRequest,
    AccountantWithRateProviders accountant,
    uint256 discount
) external nonReentrant requiresAuth {
    // Validate amount.
    uint256 offerBalance = offer.balanceOf(msg.sender);
    if (userRequest.offerAmount > offerBalance) {
        revert
AtomicQueue__SafeRequestOfferAmountGreaterThanOfferBalance(userRequest.offerAmount,
offerBalance);
    }
    // Validate deadline.
    if (block.timestamp > userRequest.deadline) {
        revert AtomicQueue__SafeRequestDeadlineExceeded(userRequest.deadline);
    }
    // Validate approval.
    uint256 offerAllowance = offer.allowance(msg.sender, address(this));
    if (offerAllowance < userRequest.offerAmount) {
        revert
```

```
AtomicQueue__SafeRequestInsufficientOfferAllowance(userRequest.offerAmount,
offerAllowance);
    }
    // Validate offerAmount is nonzero.
    if (userRequest.offerAmount == 0) revert
AtomicQueue__SafeRequestOfferAmountZero();
    // Calculate atomic price.
    if (discount > MAX_DISCOUNT) revert AtomicQueue__SafeRequestDiscountTooLarge();
    if (address(offer) != address(accountant.vault())) revert
AtomicQueue__SafeRequestAccountantOfferMismatch();
    uint256 safeRate = accountant.getRateInQuoteSafe(want);
    uint256 safeAtomicPrice = safeRate.mulDivDown(1e6 - discount, 1e6);
    if (safeAtomicPrice > type(uint88).max) revert
AtomicQueue__SafeRequestCannotCastToUint88();
    userRequest.atomicPrice = uint88(safeAtomicPrice);
    _updateAtomicRequest(offer, want, userRequest);
}
```

Solution

Add an allowlist validation for `accountant`.

Status

Acknowledged

[N3] [Suggestion] Missing Zero-Address Validation

Category: Others

Content

In `AccountantWithRateProviders`, the `constructor` and `updatePayoutAddress` functions do not check whether `payoutAddress` is the zero address.

- src/base/Roles/AccountantWithRateProviders.sol#L114-L143, L216-L219

```
constructor(
    address _owner,
    address _vault,
```

```
    address payoutAddress,
    uint96 startingExchangeRate,
    address _base,
    uint16 allowedExchangeRateChangeUpper,
    uint16 allowedExchangeRateChangeLower,
    uint32 minimumUpdateDelayInSeconds,
    uint16 managementFee
)
    Auth(_owner, Authority(address(0)))
{
    base = ERC20(_base);
    decimals = ERC20(_base).decimals();
    vault = BoringVault(payable(_vault));
    ONE_SHARE = 10 ** vault.decimals();
    accountantState = AccountantState({
        payoutAddress: payoutAddress,
        feesOwedInBase: 0,
        totalSharesLastUpdate: uint128(vault.totalSupply()),
        exchangeRate: startingExchangeRate,
        allowedExchangeRateChangeUpper: allowedExchangeRateChangeUpper,
        allowedExchangeRateChangeLower: allowedExchangeRateChangeLower,
        lastUpdateTimestamp: uint64(block.timestamp),
        isPaused: false,
        minimumUpdateDelayInSeconds: minimumUpdateDelayInSeconds,
        managementFee: managementFee
    });
}

function updatePayoutAddress(address payoutAddress) external requiresAuth {
    address oldPayout = accountantState.payoutAddress;
    accountantState.payoutAddress = payoutAddress;
    emit PayoutAddressUpdated(oldPayout, payoutAddress);
}
```

Solution

Add zero-address validation.

Status

Fixed

[N4] [High] Incorrect ZkLighter Public-Key Length Check Makes the Emergency Key-Rotation Path Unusable

Category: Design Logic Audit

Content

In `EmergencyZkLighterDecoderAndSanitizer`, `changePubKey` only allows `_pubKey` to have length `0`, while the on-chain `AdditionalZkLighter.changePubKey()` requires `_pubKey` to be 40 bytes and non-zero. As a result, calldata allowed by the emergency decoder reverts in the real ZkLighter contract, while the 40-byte non-zero public key accepted by ZkLighter is rejected by the decoder. This makes the emergency `changePubKey` path unusable in practice.

- src/base/DecodersAndSanitizers/hyperwave/aprimeusd/EmergencyZkLighterDecoderAndSanitizer.sol#L21-L31

```
// @desc Rotate the signing key for the vault's ZkLighter account.
function changePubKey(
    uint48 /*_accountIndex*/,
    uint8 /*_apiKeyIndex*/,
    bytes calldata _pubKey
) external pure returns (bytes memory addressesFound) {
    // Only allow zeroing out the pubkey to remove API key for emergency case
    if (_pubKey.length != 0) {
        revert EmergencyZkLighterDecoderAndSanitizer__InvalidPubKey();
    }
    return addressesFound;
}
```

Solution

ZkLighter does not provide an interface to remove a key. Change the function objective from disabling/removing a

key to emergency replacement with a safe public key. The decoder should require `_pubKey.length == 40` and reject an all-zero public key.

Status

Fixed

[N5] [Low] Misspelled `flashLoan` Function Signature in the Merkle Leaf Generator Invalidates Proofs

Category: Design Logic Audit

Content

In `BaseMerkleRootGenerator._addBalancerFlashloanLeafs`, the function signature used to generate the Balancer flash loan Merkle leaf is incorrectly written as `lashLoan(address,address[],uint256[],bytes)`, while the correct signature is `flashLoan(address,address[],uint256[],bytes)`. Because the Merkle leaf includes the selector derived from the function signature, this typo makes the generated selector differ from the real selector of `ManagerWithMerkleVerification.flashLoan(address,address[],uint256[],bytes)`. As a result, Balancer flash loan permission leaves generated by this helper cannot pass `_verifyCallData`, and the intended flash loan management call reverts during the first Merkle verification stage.

- resources/BaseMerkleRootGenerator.sol#L1280-L1291

```
function _addBalancerFlashloanLeafs(ManageLeaf[] memory leafs, address
tokenToFlashloan) internal {
    leafIndex++;
    leafs[leafIndex] = ManageLeaf(
        _managerAddress,
        false,
        "lashLoan(address,address[],uint256[],bytes)",
        new address[](2),
        string.concat("Flashloan ", ERC20(tokenToFlashloan).symbol(), " from Balancer
Vault"),
        _rawDataDecoderAndSanitizer
    );
    leafs[leafIndex].argumentAddresses[0] = _managerAddress;
```

```
    leafs[leafIndex].argumentAddresses[1] = tokenToFlashloan;
}
```

Solution

Change the function signature in `BaseMerkleRootGenerator._addBalancerFlashloanLeafs` from `lashLoan(address,address[],uint256[],bytes)` to `flashLoan(address,address[],uint256[],bytes)`.

Status

Fixed

[N6] [Suggestion] Missing Events for Important Parameter Updates

Category: Others

Content

1.In `RedstoneStablecoinRateProvider`, `setLowerBound`, `setMaxRate`, `setSequencerDowntimeLimit`, and `updateSequencerStatus` can modify important contract parameters but do not emit events.

2.In `TellerWithMultiAssetSupport`, `setShareLockPeriod` can modify an important contract parameter but does not emit an event.

- src/oracles/RedstoneStablecoinRateProvider.sol#L154-L183

```
function setLowerBound(uint16 _lowerBoundInBps) external onlyRole(GOVERNANCE_ROLE) {
    require(_lowerBoundInBps < 10_000, "Lower bound must be less than 100%");
    lowerBound = 10 ** RATE_DECIMALS * _lowerBoundInBps / 10_000;
}

function setMaxRate(uint16 _maxRateInBps) external onlyRole(GOVERNANCE_ROLE) {
    require(_maxRateInBps < 11_000, "Max rate must be less than 110%");
    MAX_RATE = 10 ** RATE_DECIMALS * _maxRateInBps / 10_000;
}

function setSequencerDowntimeLimit(uint64 _sequencerDowntimeLimit) external
onlyRole(GOVERNANCE_ROLE) {
```

```
    sequencerDowntimeLimit = _sequencerDowntimeLimit;
}

function updateSequencerStatus(bool isDown, uint256 startedAt) external
onlyRole(GUARDIAN_ROLE) {
    isSequencerDown = isDown;
    sequencerStartedAt = startedAt;
}
```

- src/base/Roles/TellerWithMultiAssetSupport.sol#L179-L182

```
function setShareLockPeriod(uint64 _shareLockPeriod) external requiresAuth {
    if (_shareLockPeriod > MAX_SHARE_LOCK_PERIOD) revert
TellerWithMultiAssetSupport__ShareLockPeriodTooLong();
    shareLockPeriod = _shareLockPeriod;
}
```

Solution

Add event emission for these parameter updates.

Status

Fixed

[N7] [Suggestion] Redundant Code

Category: Others

Content

The `ONE` parameter is not used in the `RedstoneStablecoinRateProvider` contract.

- src/oracles/RedstoneStablecoinRateProvider.sol#L91

```
uint256 public immutable ONE;
```

Solution

It is recommended to remove redundant code.

Status

Fixed

[N8] [Low] `cancelTransaction` Does Not Check Whether the Transaction Is Queued

Category: Design Logic Audit

Content

In `Timelock`, `cancelTransaction` does not check whether `queuedTransactions[txHash]` is `true`, which may allow canceling a transaction that was never queued.

- src/timelock/Timelock.sol#L139-L156

```
function cancelTransaction(
    address target,
    uint value,
    string memory signature,
    bytes memory data,
    uint eta
) public {
    require(
        msg.sender == admin || isCanceller[msg.sender],
        "Timelock::cancelTransaction: Call must come from admin or canceller."
    );

    bytes32 txHash = keccak256(
        abi.encode(target, value, signature, data, eta)
    );
    queuedTransactions[txHash] = false;

    emit CancelTransaction(txHash, target, value, signature, data, eta);
}
```

Solution

Check whether the specified `txHash` is actually queued.

Status

Fixed

[N9] [Medium] `ManagerWithMerkleVerification` Flash Loan Callback Reuses a Shared Root and May Bypass Strategist Permission Isolation

Category: Design Logic Audit

Content

`ManagerWithMerkleVerification` uses `manageRoot[msg.sender]` to maintain an independent Merkle root for each strategist. Although the outer `flashLoan` call must first pass verification against the initiating strategist's own root, `receiveFlashLoan` performs callback operations through `ManagerWithMerkleVerification(address(this)).manageVaultWithMerkleVerification(...)`. At that point, `msg.sender == address(this)`, so the second verification uses `manageRoot[address(manager)]` instead of the original strategist's root. If `address(manager)` is configured with a shared callback root that contains leaves with higher permissions than the initiating strategist's root, a lower-privileged strategist may craft callback `userData` to execute operations beyond its own root during the flash loan callback.

- src/base/Roles/ManagerWithMerkleVerification.sol#L99-L102, L225-L227

```
function setManageRoot(address strategist, bytes32 _manageRoot) external requiresAuth
{
    bytes32 oldRoot = manageRoot[strategist];
    manageRoot[strategist] = _manageRoot;
    emit ManageRootUpdated(strategist, oldRoot, _manageRoot);
}

ManagerWithMerkleVerification(address(this)).manageVaultWithMerkleVerification(
```

```
    manageProofs, decodersAndSanitizers, targets, data, values
);
```

Solution

Record the real initiating strategist at the `flashLoan` entry point, for example with `transient address flashLoanInitiator = msg.sender`, and use `manageRoot[flashLoanInitiator]` for the nested verification in `receiveFlashLoan` so flash-loan operations remain constrained by the initiator's own root.

Status

Fixed

[N10] [Medium] Timelock Canceller Role Can Permanently Veto Governance and Cannot Be Forcibly Removed

Category: Authority Control Vulnerability Audit

Content

In `Timelock`, `cancelTransaction` allows `admin || isCanceller[msg.sender]` to cancel any queued transaction. However, `setCanceller` requires `msg.sender == address(this)`, meaning it can only be executed through the timelock queue. A transaction to remove a canceller must therefore also wait in the queue, and a malicious or compromised canceller can cancel that transaction before it executes, as well as cancel all other queued transactions. Once a canceller is compromised, the admin can never remove it, causing permanent governance DoS and an unbreakable deadlock.

- src/timelock/Timelock.sol#L104-L111, L139-L156

```
function setCanceller(address canceller, bool enabled) public {
    require(
        msg.sender == address(this),
        "Timelock::setCanceller: Call must come from Timelock."
    );
    isCanceller[canceller] = enabled;
```

```
    emit CancellerSet(canceller, enabled);
}

function cancelTransaction(
    address target,
    uint value,
    string memory signature,
    bytes memory data,
    uint eta
) public {
    require(
        msg.sender == admin || isCanceller[msg.sender],
        "Timelock::cancelTransaction: Call must come from admin or canceller."
    );

    bytes32 txHash = keccak256(
        abi.encode(target, value, signature, data, eta)
    );
    queuedTransactions[txHash] = false;

    emit CancelTransaction(txHash, target, value, signature, data, eta);
}
```

Solution

Allow the admin to remove a canceller directly without going through the queue; or prevent cancellers from canceling queued transactions that call `setCanceller`; or introduce an emergency multisig governance path.

Status

Fixed

[N11] [Information] Cross-Chain Deposit Flow Depends on a Zero Share-Lock Period Configuration

Category: Design Logic Audit

Content

`MultiChainLayerZeroTellerWithMultiAssetSupport` inherits the share-lock mechanism from

`TellerWithMultiAssetSupport`. If `shareLockPeriod` is set to a non-zero value, `depositAndBridge()` becomes unusable because it calls `bridge()` immediately after deposit, and `bridge()` fails the `beforeTransfer(msg.sender)` check.

- src/base/Roles/CrossChain/CrossChainTellerBase.sol#L53-L55, L70-L90

```
uint256 shareAmount = _erc20Deposit(depositAsset, depositAmount, minimumMint,
msg.sender);
_afterPublicDeposit(msg.sender, depositAsset, depositAmount, shareAmount,
shareLockPeriod);
bridge(shareAmount, data);

function bridge(
    uint256 shareAmount,
    BridgeData calldata data
)
    public
    payable
    requiresAuth
    returns (bytes32 messageId)
{
    if (isPaused) revert TellerWithMultiAssetSupport__Paused();

    _beforeBridge(data);

    // Since shares are directly burned, call `beforeTransfer` to enforce before
transfer hooks.
    beforeTransfer(msg.sender);

    // Burn shares from sender
    vault.exit(address(0), ERC20(address(0)), 0, msg.sender, shareAmount);

    messageId = _bridge(shareAmount, data);
    _afterBridge(shareAmount, data, messageId);
}
```

- src/base/Roles/TellerWithMultiAssetSupport.sol#L189-L191

```
function beforeTransfer(address from) public view {
    if (shareUnlockTime[from] > block.timestamp) revert
TellerWithMultiAssetSupport__SharesAreLocked();
}
```

Solution

For cross-chain Teller deployments, keep `shareLockPeriod` fixed at `0`. If a non-zero share lock is required, redesign `depositAndBridge()` so bridging occurs only after the lock period has expired.

Status

Acknowledged

[N12] [Suggestion] Missing Decimal Compatibility Check

Category: Design Logic Audit

Content

In `RedstoneStablecoinRateProvider`, `_calculateScale()` does not validate `quoteFeedDecimals <= rateDecimals + baseFeedDecimals`. When `quoteFeedDecimals` is greater than the sum of the other two values, `10 ** (rateDecimals + baseDecimals) / 10 ** quoteDecimals` rounds down to `0` due to integer division, permanently setting the immutable `SCALE` to `0`. After that, `_calculateRate()` always computes a `rate` of `0`, preventing `getRate()` from returning a valid price and potentially blocking related deposit, withdrawal, or valuation logic. The condition can only be fixed by redeployment.

- src/oracles/RedstoneStablecoinRateProvider.sol#L202-L211

```
// Check for decimal overflow
if (rateDecimals > MAX_DECIMALS || baseDecimals > MAX_DECIMALS || quoteDecimals >
MAX_DECIMALS) {
    revert InvalidPriceFeedDecimals(MAX_DECIMALS);
}
```

```
// Calculate intermediate values to avoid overflow
uint256 numerator = 10 ** (rateDecimals + baseDecimals);
uint256 denominator = 10 ** quoteDecimals;

return numerator / denominator;
```

Solution

Add a decimal compatibility check in `_calculateScale()` to ensure that `quoteDecimals <= rateDecimals + baseDecimals`.

Status

Fixed

[N13] [Information] `AtomicQueue` Solvability Checks Ignore Locked Vault Shares

Category: Design Logic Audit

Content

`AtomicQueue` solvability checks do not account for the transfer-lock state of `BoringVault` shares. If the offer token is a `BoringVault` share and the shares are still restricted by `TellerWithMultiAssetSupport.shareUnlockTime`, `isAtomicRequestValid()` / `viewVerboseSolveMetaData()` may still show that the request can be processed. However, the actual `solve()` call reverts when `offer.safeTransferFrom(user, solver, amount)` triggers Teller's `beforeTransfer()` check, causing the solver transaction to fail and lose gas. This mainly creates solver misdirection and gas-griefing risk.

- src/atomic-queue/AtomicQueue.sol#L253-L271, L347-L378

```
function isAtomicRequestValid(ERC20 offer, address user, AtomicRequest calldata
userRequest)
    external
    view
    returns (bool)
```

```
{
    // Validate amount.
    if (userRequest.offerAmount > offer.balanceOf(user)) return false;
    // Validate deadline.
    if (block.timestamp > userRequest.deadline) return false;
    // Validate approval.
    if (offer.allowance(user, address(this)) < userRequest.offerAmount) return false;
    // Validate offerAmount is nonzero.
    if (userRequest.offerAmount == 0) return false;
    // Validate atomicPrice is nonzero.
    if (userRequest.atomicPrice == 0) return false;
    // Validate minimum request age for withdrawal.
    if (block.timestamp < userRequest.updateTime + minimumRequestAge) return false;

    return true;
}

function solve(ERC20 offer, ERC20 want, address[] calldata users, bytes calldata
runData, address solver)
    external
    nonReentrant
    requiresAuth
{
    if (isPaused) revert AtomicQueue__Paused();

    // Save offer asset decimals.
    uint8 offerDecimals = offer.decimals();

    uint256 assetsToOffer;
    uint256 assetsForWant;
    for (uint256 i; i < users.length; ++i) {
        AtomicRequest storage request = userAtomicRequest[users[i]][offer][want];

        if (request.inSolve) revert AtomicQueue__UserRepeated(users[i]);
        if (block.timestamp > request.deadline) revert
AtomicQueue__RequestDeadlineExceeded(users[i]);
        if (request.offerAmount == 0) revert AtomicQueue__ZeroOfferAmount(users[i]);
         // Check if minimum request age has passed
```

```
        if (block.timestamp < request.updateTime + minimumRequestAge) {
            revert AtomicQueue__MinimumRequestAgeNotMet(users[i]);
        }

        // User gets whatever their atomic price * offerAmount is.
        assetsForWant += _calculateAssetAmount(request.offerAmount,
request.atomicPrice, offerDecimals);

        // If all checks above passed, the users request is valid and should be
fulfilled.
        assetsToOffer += request.offerAmount;
        request.inSolve = true;
        // Transfer shares from user to solver.
        offer.safeTransferFrom(users[i], solver, request.offerAmount);
    }
```

- src/base/Roles/TellerWithMultiAssetSupport.sol#L189-L191

```
function beforeTransfer(address from) public view {
    if (shareUnlockTime[from] > block.timestamp) revert
TellerWithMultiAssetSupport__SharesAreLocked();
}
```

Solution

Synchronize `AtomicQueue` solvability checks with the `BoringVault` share-lock state.

Status

Acknowledged; The project team stated that the solvability check is performed off-chain. Before submitting a transaction, the solving bot will perform a simulation, and if any errors occur, the transaction will be aborted.

[N14] [Medium] Risk of Excessive Privilege

Category: Authority Control Vulnerability Audit

Content

1.In `BoringVault`, `setBeforeTransferHook` allows authorized roles to replace or disable the pre-transfer check logic for BoringVault shares.

2.In `TellerWithMultiAssetSupport`, authorized roles can control the supported asset list, public deposit switch, share-lock period, deposit refunds, and bulk deposit/withdraw paths.

3.In `AccountantWithRateProviders`, authorized roles can control BoringVault's core valuation parameters, exchange-rate update bounds, management fee, fee payout address, and asset rate-provider configuration.
In RedstoneStablecoinRateProvider, authorized roles can adjust price bounds and the sequencer recovery window, while the Guardian role can manually update the sequencer status.

4.In `KyberswapDecoderAndSanitizer`, `GUARDIAN_ROLE` can modify the maximum slippage protection for Kyberswap swaps.

5.In `ManagerWithMerkleVerification`, authorized roles can control each strategist's Merkle permission root.

6.In `MultiChainTellerBase`, authorized roles can control chain-level send/receive allowlists, target Teller addresses, and message gas bounds for the cross-chain Teller.

7.In `Configuration`, authorized roles can write or overwrite arbitrary aprimeUSD configuration entries.

- src/base/BoringVault.sol#L142

```
function setBeforeTransferHook(address _hook) external requiresAuth
```

- src/base/Roles/TellerWithMultiAssetSupport.sol#L133-L326

```
function pause() external requiresAuth
function unpause() external requiresAuth
function addAsset(ERC20 asset) external requiresAuth
function removeAsset(ERC20 asset) external requiresAuth
function setShareLockPeriod(uint64 _shareLockPeriod) external requiresAuth
function refundDeposit(
    uint256 nonce,
```

```
    address receiver,
    address depositAsset,
    uint256 depositAmount,
    uint256 shareAmount,
    uint256 depositTimestamp,
    uint256 shareLockUpPeriodAtTimeOfDeposit
) external requiresAuth
function deposit(ERC20 depositAsset, uint256 depositAmount, uint256 minimumMint)
    external
    requiresAuth
    nonReentrant
    returns (uint256 shares)
function depositWithPermit(
    ERC20 depositAsset,
    uint256 depositAmount,
    uint256 minimumMint,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external requiresAuth nonReentrant returns (uint256 shares)
function bulkDeposit(ERC20 depositAsset, uint256 depositAmount, uint256 minimumMint,
address to)
    external
    requiresAuth
    nonReentrant
    returns (uint256 shares)
function bulkWithdraw(ERC20 withdrawAsset, uint256 shareAmount, uint256 minimumAssets,
address to)
    external
    requiresAuth
    returns (uint256 assetsOut)
```

- src/base/Roles/AccountantWithRateProviders.sol#L151-L244

```
function pause() external requiresAuth
function unpause() external requiresAuth
```

```
function updateDelay(uint32 minimumUpdateDelayInSeconds) external requiresAuth
function updateUpper(uint16 allowedExchangeRateChangeUpper) external requiresAuth
function updateLower(uint16 allowedExchangeRateChangeLower) external requiresAuth
function updateManagementFee(uint16 managementFee) external requiresAuth
function updatePayoutAddress(address payoutAddress) external requiresAuth
function setRateProviderData(ERC20 asset, bool isPeggedToBase, address rateProvider)
external requiresAuth
function updateExchangeRate(uint96 newExchangeRate) external requiresAuth
```

- src/oracles/RedstoneStablecoinRateProvider.sol#L154-L180

```
function setLowerBound(uint16 _lowerBoundInBps) external onlyRole(GOVERNANCE_ROLE)
function setMaxRate(uint16 _maxRateInBps) external onlyRole(GOVERNANCE_ROLE)
function setSequencerDowntimeLimit(uint64 _sequencerDowntimeLimit) external
onlyRole(GOVERNANCE_ROLE)
function updateSequencerStatus(bool isDown, uint256 startedAt) external
onlyRole(GUARDIAN_ROLE)
```

- src/base/DecodersAndSanitizers/hyperwave/common/KyberswapDecoderAndSanitizer.sol#L107-L109

```
function setMaxSlippage(uint256 _maxSlippage) external onlyRole(GUARDIAN_ROLE)
```

- src/base/Roles/ManagerWithMerkleVerification.sol#L99

```
function setManageRoot(address strategist, bytes32 _manageRoot) external requiresAuth
```

- src/base/Roles/CrossChain/MultiChainTellerBase.sol#L68-L162

```
function addChain(
    uint32 chainSelector,
    bool allowMessagesFrom,
    bool allowMessagesTo,
    address targetTeller,
    uint64 messageGasLimit,
    uint64 messageGasMin
```

```
) external requiresAuth
function stopMessagesFromChain(uint32 chainSelector) external requiresAuth
function allowMessagesFromChain(uint32 chainSelector, address targetTeller) external
requiresAuth
function removeChain(uint32 chainSelector) external requiresAuth
function allowMessagesToChain(uint32 chainSelector, address targetTeller, uint64
messageGasLimit)
    external
    requiresAuth
function stopMessagesToChain(uint32 chainSelector) external requiresAuth
function setChainGasLimit(uint32 chainSelector, uint64 messageGasLimit) external
requiresAuth
```

- src/base/aprimeUSD/Configuration.sol#L24-L27

```
function setConfiguration(string calldata key, string calldata value) external
requiresAuth
```

Solution

In the short term, transferring the ownership of core roles to time-locked contracts and managing them by multi-signatures is an effective solution to avoid single-point risks. However, in the long run, a more reasonable solution is to implement a permission separation strategy and set up multiple privileged roles to manage each privileged function separately. Permissions involving user funds and contract core parameter updates should be managed by the community, while permissions involving emergency contract suspensions can be managed by EOA addresses. This ensures the safety of user funds while responding quickly to threats.

Status

Acknowledged

[N15] [Low] `MultiChainTellerBase` Missing `minimumMessageGas <= messageGasLimit` Validation

Category: Design Logic Audit

Content

In the `MultiChainTellerBase` contract, `addChain` , `allowMessagesToChainand` and `setChainGasLimit` do not validate that minimumMessageGas <= messageGasLimit. This can create a dead configuration where min > max, causing all bridge() requests for that chain to continuously revert.

- src/base/Roles/CrossChain/MultiChainTellerBase.sol:L68-L86, L128-L145, L162-L170

```
function addChain(
    uint32 chainSelector,
    bool allowMessagesFrom,
    bool allowMessagesTo,
    address targetTeller,
    uint64 messageGasLimit,
    uint64 messageGasMin
)
    external
    requiresAuth
{
    if (allowMessagesTo && messageGasLimit == 0) {
        revert MultiChainTellerBase_ZeroMessageGasLimit();
    }
    selectorToChains[chainSelector] =
        Chain(allowMessagesFrom, allowMessagesTo, targetTeller, messageGasLimit,
messageGasMin);

    emit ChainAdded(chainSelector, allowMessagesFrom, allowMessagesTo, targetTeller,
messageGasLimit, messageGasMin);
}

function allowMessagesToChain(
        uint32 chainSelector,
        address targetTeller,
        uint64 messageGasLimit
    )
        external
        requiresAuth
    {
```

```
        if (messageGasLimit == 0) {
            revert MultiChainTellerBase_ZeroMessageGasLimit();
        }
        Chain storage chain = selectorToChains[chainSelector];
        chain.allowMessagesTo = true;
        chain.targetTeller = targetTeller;
        chain.messageGasLimit = messageGasLimit;

        emit ChainAllowMessagesTo(chainSelector, targetTeller);
    }

function setChainGasLimit(uint32 chainSelector, uint64 messageGasLimit) external
requiresAuth {
    if (messageGasLimit == 0) {
        revert MultiChainTellerBase_ZeroMessageGasLimit();
    }
    Chain storage chain = selectorToChains[chainSelector];
    chain.messageGasLimit = messageGasLimit;

    emit ChainSetGasLimit(chainSelector, messageGasLimit);
}
```

Solution

Add a minimumMessageGas <= messageGasLimit validation in addChain and setChainGasLimit.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002607020002	SlowMist Security Team	2026.06.23 - 2026.07.02	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 3 medium risks, 3 low risks, 6 suggestions, 2 informationals vulnerabilities.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.



Official Website

www.slowmist.com



E-mail

team@slowmist.com



Twitter

@SlowMist_Team



Github

https://github.com/slowmist