> security_review · cantina_bug_bounty

AggLayer Vault Bridge

Independent Security Assessment — Full Audit Report

PROTOCOL	PLATFORM	DATE
AggLayer Vault Bridge	Cantina	May 2026

REPOSITORY	LINES REVIEWED
agglayer/vault-bridge	3,039

HIGH	MEDIUM	LOW	INFORMATIONAL	TOTAL
2	3	5	3	13

AUDITOR

Prameya Singh Soni

blackburn

// Executive Summary

This report presents the findings of an independent security assessment of the Agglayer Vault Bridge codebase. The audit covered all primary-chain contracts and the secondary-chain NativeConverter, spanning 13 audit passes: access control, state invariants, external interactions, economic attacks, edge cases, known vulnerability patterns, upgradeability, cross-chain message validation, signature security, accounting precision, and trust assumption analysis.

The audit identified **2 High, 3 Medium, 5 Low, and 3 Informational** severity findings, covering broken cross-chain redemption, unlimited token approvals to arbitrary addresses, missing timelocks, yield oracle manipulation, CEI violations, and documentation gaps.

The codebase demonstrates a mature engineering approach — ERC-7201 namespaced storage, transient reentrancy guards, try/catch deposit patterns, and thorough use of OZ AccessControl. The primary concerns are architectural: a core user-facing function that is permanently broken, instant unlimited approval grants with no delay, and a complete absence of timelocks on fund-affecting governance functions.

// Scope

FILE	LINES
src/primary-chain/VaultBridgeToken.sol	1,359
src/primary-chain/VaultBridgeTokenPart2.sol	381
src/primary-chain/VaultBridgeTokenInitializer.sol	117
src/primary-chain/MigrationManager.sol	339
src/primary-chain/ethereum/vbETH/VbETH.sol	132
src/secondary-chain/NativeConverter.sol	658
src/etc/IAgglayerBridge.sol	53
Total	**3,039**

// Findings Summary

ID	TITLE	SEVERITY	STATUS
H-01	Unprotected reinitialize(bytes[]) enables upgrade front-running and proxy takeover	HIGH	Open
H-02	setYieldVault grants type(uint256).max approval to arbitrary address instantly	HIGH	Open
M-01	claimAndRedeem always reverts — cross-chain redemption flow permanently broken	MEDIUM	Open
M-02	No timelock on any critical governance function — single-block admin drain possible	MEDIUM	Open
M-03	Yield vault share-price oracle manipulation enables vbToken over-minting and insolvency	MEDIUM	Open
L-01	CEI violation: _burn executes after yieldVault.withdraw external call in _withdraw	LOW	Open
L-02	burn() reverts when yieldRecipient holds vbToken from sources other than collectYield	LOW	Open
L-03	Withdrawal auto-rebalance silently disabled when minimumReservePercentage < 10%	LOW	Open
L-04	setCustomToken permanently blocked if secondary-chain migration bridge claim is never executed	LOW	Open
L-05	migrationFeesFund has no withdrawal path — excess donations permanently locked	LOW	Open
I-01	_depositIntoYieldVault catch block assumes fixed revert data format	INFO	Open
I-02	setYieldRecipient NatSpec inaccurately guarantees pre-collection when precollectYield=false	INFO	Open
I-03	Directly transferred underlying tokens bypass accounting and are permanently unrecoverable	INFO	Open

// Detailed Findings

H-01

Unprotected reinitialize(bytes[]) Enables Upgrade Front-Running and Proxy Takeover

HIGH

reinitialize(bytes[]) is a public function with no access control. Any address can call it. The function uses globalInitializationCounter to determine which pending reinitializer steps to execute, then delegatecalls them on the implementation with caller-supplied data.

Attack vector 1 — Upgrade front-running: When a new reinitializer step is added during an upgrade, an attacker monitors the mempool, sees the pending reinitialize call, and front-runs with malicious data. If the reinitializer sets critical state (roles, addresses, configuration), the attacker controls those values.

Attack vector 2 — Fresh proxy race window: If proxy deployment and initialization are two separate transactions, an attacker calls reinitialize([maliciousData1, maliciousData2]) before the deployer, granting themselves DEFAULT_ADMIN_ROLE.

The locked() modifier in InitializationCounterUpgradeable was designed to restrict reinitializers but is never applied to reinitialize1 or reinitialize2 in any contract — it is dead code.

Impact: A successfully front-run initialization grants the attacker DEFAULT_ADMIN_ROLE, allowing instant drain of all protocol funds via setYieldVault, drainYieldVault, and setYieldRecipient.

Recommendation: Add onlyRole(DEFAULT_ADMIN_ROLE) to reinitialize(bytes[]) in all contracts. Alternatively, use an atomic deploy-and-initialize pattern (CREATE2 with initialization calldata) to eliminate the race window.

CONTRACTS	LIKELIHOOD	IMPACT
VbETH.sol, VaultBridgeToken.sol, MigrationManager.sol	Medium	Critical

H-02

setYieldVault Grants type(uint256).max Approval to Arbitrary Address Instantly

HIGH

When the yield vault is changed, setYieldVault immediately grants type(uint256).max approval of the underlying token to the new vault address with no verification and no delay. yieldVault_ is only checked for non-zero — it is not required to implement IERC4626 and can be any address including an EOA.

```
function setYieldVault(address yieldVault_) external override
onlyRole(DEFAULT_ADMIN_ROLE) nonReentrant {
    // ...
    $.underlyingToken.forceApprove(address($.yieldVault), 0);
    $.yieldVault = IERC4626(yieldVault_);
    $.underlyingToken.forceApprove(yieldVault_, type(uint256).max); // ←
unlimited approval, no validation
    emit YieldVaultSet(yieldVault_);
}
```

Attack chain: (1) Attacker compromises the DEFAULT_ADMIN_ROLE key. (2) Calls setYieldVault(attackerAddress) — unlimited approval granted in the same transaction. (3) Calls underlyingToken.transferFrom(vbToken, attacker, balance) — drains all reserve assets. Combined with drainYieldVault(), 100% of TVL across all vbToken instances (vbUSDC, vbUSDT, vbWBTC, vbUSDS, vbETH) is drainable in one block.

Recommendation: Implement a minimum 48–72 hour timelock on setYieldVault. Add an interface check before approving: verify IERC4626(yieldVault_).asset() == underlyingToken. At minimum, require a separate restoreYieldVaultApproval() call after a delay.

CONTRACT	LIKELIHOOD	IMPACT
VaultBridgeTokenPart2.sol:309–326	Low	Critical

M-01

claimAndRedeem Always Reverts — Cross-Chain Redemption Flow Permanently Broken

MEDIUM

claimAndRedeem() is the designated atomic function for users bridging vbTokens from a secondary chain back to the primary chain and redeeming underlying assets. The function always reverts for all real-network callers.

When bridging back, the leaf is constructed with destinationAddress: address(vbToken) — tokens are delivered to the vbToken contract itself. The contract then calls _withdraw, which calls _spendAllowance(address(vbToken), msg.sender, shares). This checks whether the external caller has been approved by the vbToken contract to spend its own balance — an approval that is never set anywhere in the codebase. The call reverts with ERC20InsufficientAllowance on 100% of invocations.

The protocol's own integration test works around this bug using vm.prank(address(vbToken)) — a Foundry cheat code with no EVM equivalent on production networks.

Impact: Any user who bridges vbTokens using the protocol-designed leaf will have their vbTokens delivered to the contract with no way to redeem the underlying. The Agglayer bridge leaf is immutable once submitted — there is no on-chain recovery path short of an admin upgrade.

Recommendation: After claimAsset delivers vbTokens to address(this), grant the caller a temporary self-approval before calling _withdraw. The allowance is consumed atomically by _spendAllowance.

CONTRACT	LIKELIHOOD	IMPACT
VaultBridgeToken.sol:789–826	High	High

M-02

No Timelock on Any Critical Governance Function — Single-Block Admin Drain Possible

MEDIUM

The entire vault-bridge codebase contains zero timelock logic. All fund-affecting governance functions execute instantly in the same block as the admin call: setYieldVault, drainYieldVault, setYieldRecipient, setMinimumReservePercentage, and configureNativeConverters.

Combined with H-02, a single compromised DEFAULT_ADMIN_ROLE key can drain 100% of TVL across all vbToken contracts in one block with no on-chain warning or recovery window for users.

Recommendation: Wrap fund-affecting setters behind a TimelockController (OpenZeppelin) with a minimum 48–72 hour delay. Exempt emergency functions (pause, revokeYieldVaultApproval) from the timelock to allow fast incident response.

CONTRACTS	LIKELIHOOD	IMPACT
VaultBridgeTokenPart2.sol, MigrationManager.sol	Low	Critical

M-03

Yield Vault Share-Price Oracle Manipulation Enables vbToken Over-Minting and Insolvency

MEDIUM

collectYield() mints vbTokens to the yield recipient based on yield(), which depends on stakedAssets(). yieldVault.convertToAssets() is a real-time on-chain view that reflects the current exchange rate. For ERC-4626 vaults whose rate is based on totalAssets / totalSupply, a flash loan into the yield vault's underlying pool temporarily inflates the reported value.

```
function stakedAssets() public view returns (uint256) {
    return
$.yieldVault.convertToAssets($.yieldVault.balanceOf(address(this)));
}
```

Attack path: (1) Attacker flash loans a large amount into the yield vault's underlying pool, inflating convertToAssets(). (2) yield() returns an inflated positive amount. (3) collectYield() is triggered. (4) Excess vbToken is minted to the yield recipient. (5) Flash loan unwinds → totalAssets() < totalSupply(). (6) Protocol is insolvent — last withdrawers cannot fully redeem.

Recommendation: Document that integrated yield vaults must be flash-loan-resistant. Consider adding a per-block cap on collectYield or a cooldown between yield collection rounds.

CONTRACTS	LIKELIHOOD	IMPACT
VaultBridgeToken.sol, VaultBridgeTokenPart2.sol	Medium	High

L-01

CEI Violation: _burn Executes After yieldVault.withdraw External Call in _withdraw

LOW

In _withdraw, the share burn occurs after the yield vault external call. Between lines 712 and 730, the yield vault is executing while owner's vbToken shares remain unburned. The nonReentrant guard currently prevents exploitation, but the pattern violates CEI and creates unnecessary surface area if the reentrancy guard is ever relaxed.

```
_withdrawFromYieldVault(...)  // line 712 — external call to
$.yieldVault.withdraw()
// ...
_burn(owner, shares)          // line 730 — state change AFTER external
interaction
```

Recommendation: Move _burn(owner, shares) before the _withdrawFromYieldVault call.

CONTRACT	LIKELIHOOD	IMPACT
VaultBridgeToken.sol:712–730	Low	Low

L-02

burn() Reverts When yieldRecipient Holds vbToken From Sources Other Than collectYield

LOW

burn() decrements _netCollectedYield before calling _burn. _netCollectedYield only tracks vbTokens minted through collectYield. If the yield recipient receives vbTokens from any other source (ERC-20 transfer, airdrop, migration), calling burn(shares > _netCollectedYield) causes an arithmetic underflow and reverts. The yield recipient cannot burn legitimately held tokens that did not come from collectYield.

Recommendation: Floor _netCollectedYield at zero rather than reverting on underflow, or document clearly that burn is strictly limited to collected yield tokens.

CONTRACT	LIKELIHOOD	IMPACT
VaultBridgeTokenPart2.sol:90–104	Low	Low

L-03

Withdrawal Auto-Rebalance Silently Disabled When minimumReservePercentage < 10%

LOW

The post-withdrawal auto-rebalance trigger has an undocumented lower bound. If minimumReservePercentage is set below 10%, the third condition of the rebalance trigger ($.minimumReservePercentage >= 0.1e18) is never true — the withdrawal auto-rebalance is permanently silenced. The reserve can drain to 0% with no automatic replenishment. The setMinimumReservePercentage NatSpec does not mention this behavior.

Recommendation: Document the >= 0.1e18 lower threshold explicitly in setMinimumReservePercentage NatSpec.

CONTRACT	LIKELIHOOD	IMPACT
VaultBridgeToken.sol:739	Low	Low

L-04

setCustomToken Permanently Blocked If Secondary-Chain Migration Bridge Claim Is Never Executed

LOW

The migration flow increments _migrationsInProgressCount on migrateBackingToPrimaryChain and decrements it only when the resulting bridge-asset claim is executed on the secondary chain. setCustomToken guards on this counter. If any migration bridge-asset claim is never executed (network outage, forgotten claim, or a rogue MIGRATOR_ROLE initiating a 1-wei migration intentionally), _migrationsInProgressCount stays permanently elevated and setCustomToken can never be called again.

Recommendation: Add an admin-controlled emergency reset for _migrationsInProgressCount, or implement a timeout after which in-flight migrations can be administratively cleared.

CONTRACTS	LIKELIHOOD	IMPACT
NativeConverter.sol:567–606	Low	Medium

L-05

migrationFeesFund Has No Withdrawal Path — Excess Donations Permanently Locked

LOW

donateForCompletingMigration(uint256 assets) increases $.migrationFeesFund by assets. The fund is consumed only by completeMigration to cover fee-on-transfer discrepancies. For primary in-scope assets (USDC, USDT, WBTC, USDS), transfer fees are zero on mainnet — meaning the fund is never decremented. Any tokens donated are permanently stranded: not included in totalAssets(), not visible to yield(), and no recovery function exists.

Recommendation: Add an admin function to recover excess migrationFeesFund after migrations complete, or convert unused balance to reservedAssets.

CONTRACTS	LIKELIHOOD	IMPACT
VaultBridgeTokenPart2.sol, VaultBridgeToken.sol	Low	Low

I-01

_depositIntoYieldVault Catch Block Assumes Fixed Revert Data Format — Unexpected Panics Cause Deposit DoS

INFO

The catch block assumes that performReversibleYieldVaultDeposit always reverts with exactly abi.encode(bool, bytes, bool) format. An arithmetic underflow inside the function produces a Panic(0x11) revert — format that abi.decode cannot handle, propagating as a hard revert from the user's deposit() call. Out-of-gas scenarios produce empty revert data, causing the same failure.

Recommendation: Validate data.length before decoding, or add a catch {} fallback to handle unexpected revert formats gracefully.

CONTRACT	IMPACT
VaultBridgeToken.sol:1162–1198	Informational

I-02 setYieldRecipient NatSpec Inaccurately Guarantees Pre-Collection When precollectYield=false

INFO

The NatSpec states: "*Yield will be collected before changing the recipient.*" This is only accurate when precollectYield=true. With precollectYield=false, no collection occurs and the recipient is changed immediately. An admin calling setYieldRecipient(false, newRecipient) followed immediately by collectYield() redirects all pre-change accrued yield to the new recipient.

Recommendation: Update NatSpec to: "*Yield will be collected before changing the recipient if precollectYield is true.*"

CONTRACT
VaultBridgeToken.sol:1070

IMPACT
Informational

I-03 Directly Transferred Underlying Tokens Bypass Accounting and Are Permanently Unrecoverable

INFO

totalAssets() = stakedAssets() + reservedAssets() — it does not use underlyingToken.balanceOf(address(this)). If any underlying tokens are transferred directly to the vbToken contract (accidental transfer, rebasing edge case, or deliberate dust), they are not counted in totalAssets(), do not contribute to yield(), and no sweep function exists. They are permanently locked.

Recommendation: Add an admin sweep function that moves balanceOf(address(this)) - reservedAssets - migrationFeesFund to the yield vault or reserve.

CONTRACT
VaultBridgeToken.sol

IMPACT
Informational