Roburna Arborswap Bridge System — Comprehensive GPT Created Security Audit Report 2025-12-19.

---

1) Executive Summary

Scope

This assessment covers the on-chain components of the bridge system:

1. **BridgeCore** (flat file, includes `BridgeCore`, `FeeManager`, `MultiOwnerGovernance`, OZ deps)
2. **GeneralV2Adapter** (Uniswap V2-style swap adapter)
3. **Vault** (implementation you provided)
4. **FUSDV2** (your “stable” token contract posted here)

Key Conclusions

- **Critical correctness bug in BridgeCore refunds** for mint/burn deposits (refund tries to withdraw from Vault even though tokens were burned and never vaulted).

- **Critical design risk:** current “ValidatorManager is an EOA operator” means the bridge is effectively custodial; compromise/malice of that key can drain bridge liquidity and/or mint assets.

- **High-risk economic/design issue:** the bridge assumes a stable backing token (“USDC”) but you’ve clarified this is **FUSD**, which is **taxed** and behaves like a **fee-on-transfer token** and is **not a true stablecoin**. This undermines:

    - accounting correctness (if used as vault backing),
    - swap correctness (adapter/router methods),
    - user expectations of stable value.
- **Validation model mismatch:** A single confirmer will not scale to “multiple validators” unless you implement an on-chain **multi-confirm** system (threshold signatures or on-chain vote-based execution).

Overall Security Rating

High Risk / Not production-ready under the current architecture and token assumptions.

2) Trust Model & Threat Model

2.1 Current Trust Model (Per Your Statement)

- “Validator manager is just an address operator that confirms” (single EOA).
- “Single Gov 1/1 for now” (tests).

Implication: This is not a trustless bridge. It is a **trusted operator custody system**. Smart contracts cannot prevent malicious or compromised operator releases if the only gate is `msg.sender == validatorManager`.

2.2 Primary Threats

1. **Operator key compromise** → immediate draining of vault liquidity / arbitrary releases.
2. **Malicious operator** → censorship (refuse to release), fabricated releases (release without valid deposits).
3. **Fee-on-transfer “stable” backing** → break swap assumptions and user accounting; destabilize bridge solvency and UX.
4. **Refund/liveness failures** → users cannot recover funds after timeout.

---

3) Architecture Overview

3.1 BridgeCore (Core Bridge)

Deposit paths

- Validations: blacklist, min/max, token support.
- Generates `depositId`.
- **Mintable/Burnable path:**
    - Transfers token into BridgeCore.
    - Burns token.
    - Stores deposit info with `token = originalToken` and `amount = originalAmount`.
    - No Vault custody created.
- **Non-mintable path:**

    - Converts deposit asset into “USDC” variable (you intend FUSD) via adapter/router.
    - Deposits that token into Vault.
    - Stores deposit info with `token = usdcToken` and `amount = usdcAmount`.

Release

- `executeReleaseFromValidator(...)` is callable only by `validatorManager`.
- Marks deposits processed.
- Enforces `block.timestamp <= timestamp + 1 hour` (very strict).
- Withdraws from Vault (stored token) and optionally swaps to output token.

Refund

- After `REFUND_TIMEOUT`, user can refund unprocessed deposits.
- Refund currently always calls `vault.withdraw(depositInfo.token, depositInfo.amount, user)`.

3.2 Vault

- Custodies ERC20 supported tokens via internal accounting `tokenBalances[token]`.
- Only BridgeCore can deposit/withdraw.
- Governance can pause and emergency withdraw.

3.3 GeneralV2Adapter

- Whitelisted caller.
- Swaps via Uniswap V2 style router.
- Always routes token swaps through `wrappedToken` with a 3-hop path.
- Uses **standard** swap calls (not “supporting fee-on-transfer” variants).

3.4 FUSDV2

- ERC20 with configurable buy/sell taxes (up to 25%).
- Swap-back mechanism that sells accumulated fee tokens for ETH via Uniswap router function that assumes WETH output.
- Controllers can mint/burn.

---

4) Findings Summary (Final)

ID	Severity	Title	Component
BC-01	**Critical**	Mint/burn deposit refunds are broken: refund withdraws from Vault even though tokens were burned and never vaulted	BridgeCore
BC-02	**Critical (Design)**	Single EOA validator can unilaterally authorize releases; compromise = total loss	BridgeCore / System
BC-03	**High**	1-hour release expiry will strand legitimate releases and break liveness	BridgeCore
BC-04	**High**	Using **FUSDV2 as "stable backing"** is economically and technically unsafe (taxed token; not stable)	System
BC-05	**High**	Swap infrastructure is not compatible with fee-on-transfer tokens (adapter uses non-supporting swap methods)	Adapter / BridgeCore
BC-06	**High**	Validation system does not support multiple validators; must implement on-chain threshold/multi-confirm	System
BC-07	**Medium**	Allowance/approve patterns can fail on nonstandard ERC20s and increase allowance blast radius	BridgeCore / Adapter
BC-08	**Medium**	RefundExpiredDeposits scalability + batch fragility (O(n) scan; single revert can break batch)	BridgeCore
AD-01	**Medium**	Adapter forces wrapped-token routing; edge cases + higher slippage	Adapter
VA-01	**Medium**	Vault internal accounting can desync from real balances (direct ERC20 transfers / forced ETH)	Vault
FUSD-01	**High**	FUSD is not a stablecoin; taxes + central control + minting make it unsuitable as "stable backing"	FUSDV2
FUSD-02	**High**	Swap-back uses `swapExactTokensForETHSupportingFeeOnTransferTokens` with `baseToken` path; only valid if baseToken is WETH	FUSDV2
FUSD-03	**Medium**	Token can become non-transferable if swap-back reverts; swap is executed inside `_update`	FUSDV2
GOV-01	**Low**	Governance lacks timelock/expiry; permissive execution	MultiOwnerGovernance

5) Detailed Findings & Recommendations

BC-01 — Critical — Mint/Burn Deposit Refund Logic is Broken

What happens

- Deposit (mintable/burnable): tokens are **burned** in BridgeCore.
- Stored `DepositInfo.token` / `amount` are set to the original token/amount.
- Refund always withdraws `DepositInfo.token` from Vault — but Vault never had it.

Impact

- Refund calls revert for mint/burn deposits.
- Users can get stuck permanently if release never occurs.
- Batch refund function can be DoS'd if one refund reverts.

Required Fix

Add an explicit deposit classification and refund correctly:

Recommended structure changes

- Add to `DepositInfo`: `bool isMintable;`
- On mintable deposit: store `isMintable=true`, store original token+amount.
- On refund:
    - if `isMintable`: **mint back** to user (reverse burn) or otherwise reverse the burn mechanism.
    - else: `vault.withdraw(storedToken, storedAmount, user)`

Also: make `refundExpiredDeposits()` robust with try/catch and emit "RefundFailed" events.

BC-02 — Critical (Design) — Single EOA Validator = Total Loss on Compromise

What happens

BridgeCore trusts:

- `msg.sender == validatorManager`

If validatorManager is an EOA:

- one key can release arbitrary vault funds and/or mint flows.

Impact

- Compromise = full bridge drain.
- Malicious operator = censorship or theft.

Required Fix (Minimum)

- Replace validatorManager EOA with **multisig** immediately (2/3+).
- Add emergency pause + monitoring.

BC-03 — High — 1-Hour “Signature Validity” Window Breaks Liveness

What happens

Release requires:

- `block.timestamp <= timestamp + 1 hours`

Impact

- If ops/delays exceed 1 hour (common), releases fail permanently.
- Users are forced into refunds (which are broken for mint/burn deposits currently).

Fix

- Remove the check, or increase it to multiple days, or replace with a more meaningful replay control.
- Replay is already prevented by `processedDeposits[depositId]`.

BC-04 — High — “Stable backing token” is actually taxed & not stable (FUSDV2)

BridgeCore is designed around an internal “USDC” custody token.

- The system uses **FUSD**, claimed stable
- FUSD is “Safemoon style with taxes”

Impact on the bridge system

1. **Economic**: Not stable. Bridge collateral is volatile, taxed, and centrally controlled.

2. **Technical**:

    - Fee-on-transfer behavior can break swap and accounting assumptions.
    - Users may receive less than expected after swaps.
    - “Fee always in stable” logic becomes “fee always in taxed token”, unpredictable.

Recommendation

You should not treat FUSDV2 as a stable backing asset. Either:

- Use a true stable asset (non-taxed), or
- If chain lacks stablecoins, bridge should custody a **non-taxed canonical asset** or implement a separate hedging/pegging mechanism (off-chain / overcollateralized).

At minimum, update docs/UI and contract naming so users aren’t misled.

BC-05 — High — Swap adapter not compatible with fee-on-transfer tokens

Your adapter uses standard V2 swap methods (typical `swapExactTokensForTokens` style). These are often incompatible with taxed tokens unless “supporting fee-on-transfer” functions are used.

Impact

- Swaps involving FUSD (taxed on buys/sells via LP) can revert or deliver less than expected.
- Bridge deposits/releases that rely on swapping can fail in production.

Fix

- Add “supportingFeeOnTransferTokens” swap routes in the adapter for tokens flagged as taxed/FOT.
- Alternatively, disallow taxed tokens from swap paths entirely and require direct payout without swaps.

BC-06 — High — Multi-validator “multiple confirms” requires a new on-chain validation mechanism

You asked for a solution where:

- validator EOA needs fix
- governance/validators should provide multiple confirmations

Required Design Change

You need a **ValidatorManager contract** that supports threshold confirmation.

Two standard patterns:

Pattern A — Threshold Signatures (recommended)

- Validators sign a typed message (EIP-712) containing:
    - depositId, sourceChainId, targetChainId, token addresses, amounts, recipient, timestamp, deadline, contract address, chainId.
- BridgeCore verifies **M-of-N** signatures on-chain.
- No need for multiple on-chain transactions; one execution call includes signatures.

Pros: efficient, standard bridge pattern **Cons:** requires signature aggregation and careful implementation

Pattern B — On-chain Multi-Confirm (simple but heavier)

- `confirmRelease(depositId, paramsHash)` called by each validator.
- Store confirmations in mapping.
- When confirmations >= threshold, anyone (or a role) calls `executeRelease(...)`.

Pros: easy conceptually, easy to audit **Cons:** multiple on-chain txs; higher gas; needs careful replay/param-binding

Governance Integration

If you want governance owners to be validators:

- Use MultiOwnerGovernance owners as validator set.
- Set threshold to N (e.g., 2-of-3).
- But you must still bind the release parameters to a unique hash and prevent tampering.

Non-negotiable: BridgeCore must validate that the **same exact release parameters** were approved/signed.

---

6) Vault Audit (Implementation Provided)

Positive Properties

- `onlyBridgeCore` on `deposit()` and `withdraw()`
- `nonReentrant` and `whenNotPaused` on hot paths
- governance-controlled emergency withdraw and pause roles

Material Issues

VA-01 — Medium — Internal accounting may desync

Vault tracks `tokenBalances[token]` internally. If ERC20 is transferred directly to the vault address (not via vault functions), balances can desync. Native can be forced in via selfdestruct.

Impact: funds can be stuck from accounting perspective.

Fix: add governance-only sync methods:

- `syncERC20(token)` sets internal balance to `IERC20(token).balanceOf(address(this))`
- `syncNative()` sets internal balance to `address(this).balance`

Native handling mismatch

Vault supports native in `fundVault` and emergency native withdrawals, but normal bridge deposit/withdraw disallow native. That's fine, but document it clearly.

---

7) FUSDV2 Audit (Token Presented as “Stable”)

FUSD-01 — High — Not a stablecoin (taxed, centrally controlled, mintable)

Why

- Configurable buy/sell taxes up to **25%**
- `onlyOwner` can change taxes and wallets
- `onlyOwner` can set controllers who can **mint unlimited** supply
- Blacklist capability
- Swap-back converts fees to ETH and pays wallets → “meme token” economics, not stable mechanics

Conclusion: FUSDV2 cannot honestly be treated as a “stable backing token” for a bridge.

FUSD-02 — High — Swap-back function assumes baseToken is WETH

In `swapAndDistributeBaseToken()`:

- It builds path `[address(this), baseToken]`
- Calls `swapExactTokensForETHSupportingFeeOnTransferTokens`

That router function requires the path to end in **WETH** (because it unwraps WETH to ETH). If `baseToken` is not WETH, this will revert.

Impact: swap-back may fail permanently depending on configuration.

Fix Options:

- If baseToken is intended to be WETH, enforce it in constructor:
    - `require(baseToken_ == uniswapV2Router.WETH(), "baseToken must be WETH");`
- If baseToken is not WETH, use `swapExactTokensForTokensSupportingFeeOnTransferTokens` and then handle baseToken distribution.

FUSD-03 — Medium — Token can become non-transferable if swap-back reverts

Swap-back is called inside `_update` when `swapCondition` is true. If swap-back reverts due to:

- no liquidity,
- wrong router,
- wrong base token path,
- router failure,

then **normal token transfers revert**.

Fix:

- Make swap-back best-effort: wrap it in try/catch or add a flag to disable swap-back.
- Consider rate limiting, min amounts, and explicit admin-controlled swap trigger rather than automatic on transfer.

Additional Observations (FUSD)

- `isWhitelisted` mapping is not used for transfer restrictions—likely incomplete design.
- `liquidityInjectionThreshold` is unused—suggests unfinished “liquidity” logic.
- Approves router with `type(uint256).max`—ok if router is trusted and immutable, but still a risk surface.

---

8) System-Level Impact of Using FUSD as Bridge Custody Token

If BridgeCore's `usdcToken` is set to FUSD:

- "Fee always paid in stable" becomes "fee in taxed token"
- If BridgeCore swaps into/out of FUSD on LP pairs, taxes apply and the adapter may revert (BC-05)
- Even if Vault transfers are tax-free (non-LP), the **DEX interactions are not**
- Users may receive materially less than expected due to taxes and slippage

System recommendation: Do not use a taxed token as the canonical custody asset of a bridge.

---

9) Required Remediation Plan (Final)

Must Fix Before Any Real Usage

1. **Fix BC-01** (mint/burn refund correctness + batch refund resilience)
2. **Replace EOA validator** with multisig immediately (at least)
3. Implement a **multi-confirm validation system**:
    - EIP-712 threshold signatures (preferred) or
    - on-chain confirmations + threshold
4. Remove or extend **1-hour expiry** (BC-03)
5. Decide whether FUSD is acceptable at all as custody asset (recommend: no)
6. Update swap adapter to support fee-on-transfer swaps or disallow such tokens

Strongly Recommended

- Vault sync/reconciliation functions (VA-01)
- Preflight checks for roles/whitelists/config (to avoid bricking deposits/releases)
- Add timelocks for changing validator/vault/router/adapter addresses

---

10) “Multiple Validators” — A Working Design That Fixes Single Confirm

Here is a practical blueprint you can implement without changing your entire system:

Option 1 (Recommended): ValidatorManager with Threshold EIP-712 Signatures

- Deploy `ValidatorManager` (contract) storing:
    - `mapping(address=>bool) isValidator`
    - `uint256 threshold`
    - `uint256 nonce` or `usedReleaseHash`
- BridgeCore changes `executeReleaseFromValidator` to:
    - accept `signatures[]` and `validators[]`
    - compute `releaseHash = EIP712Hash(params…)`
    - verify `M` unique validator signatures
    - mark `releaseHash` used
    - execute release

This gives “multiple confirms” in one tx, and removes reliance on EOA operator.

Option 2: On-chain Confirmations (Governance-as-Validator)

- Add `confirmRelease(depositId, paramsHash)` callable by validators (governance owners).
- Once confirmations >= threshold, BridgeCore executes release.
- Store and check `paramsHash` to ensure all confirm the exact same payload.

This can be integrated directly with your MultiOwnerGovernance owners list.

---

11) Final Assessment

Severity Summary

- **Critical:** BC-01, BC-02
- **High:** BC-03, BC-04, BC-05, BC-06, FUSD-01, FUSD-02
- **Medium:** BC-07, BC-08, AD-01, VA-01, FUSD-03
- **Low/Info:** GOV-01 and assorted UX/semantic issues

Bottom Line

- The bridge cannot be considered secure or reliable until:
    - mint/burn refunds are fixed,
    - the validator model is upgraded to multi-confirm threshold validation,
    - and the custody “stable” asset is either a true non-taxed stablecoin or the system is redesigned to account for fee-on-transfer behavior.