Security Assessment
Final Report

Vault Bridge

May 2025

Prepared for Polygon Labs

Table of content

Project Summary **4**
- Project Scope 4
- Project Overview 4
  - Protocol Overview 5
- Findings Summary 6
- Severity Matrix 6

Detailed Findings **7**
- High Severity Issues 9
- H-01 Native tokens will be stuck in the bridge due to missing receive() function 9
- H-02 WETHNativeConverter.migrateGasBackingToLayerX() will always revert due to nonReentrant modifier in receive() 10
- Medium Severity Issues 11
- M-01 Unexpected approval set due to wrong permit 11
- M-02 claimAndRedeem() will not work if exchange ratio is ever changed due to mixup with assets and shares 13
- M-03 Deposits may fail due to Morpho vault reporting inaccurate maxDeposit() values 14
- Low Severity Issues 16
- L-01 The allocation of underlying tokens between yieldVault and reserve can be manipulated 16
- L-02 Fees tokens in combination with the yieldVault create a loss 18
- L-03 Missing __ReentrancyGuard_init() 19
- L-04 yieldVault can be accessed even when paused, which can interfere with drainVault() 20
- L-05 Denial of service for drainYieldVault() via donateAsYield() 22
- L-06 Function __Nonces_init() not called 24
- L-07 Assets from migrateBackingToLayerX() could get stuck if changes are made in the bridge 26
- L-08 Offline indexers could be confused by dual emits from _withdraw() 27
- L-09 Unnecessary complexity in USDT transfer fee calculation logic 28
- L-10 whenNotPaused modifier in admin-restricted functions may prevent recovery when contracts are paused 30
- Informational Issues 31
- I-01. Unused struct NativeConverterConfiguration 31
- I-02. Reentrancy library could also use transient storage 31
- I-03. bridgeAsset() of native asset (ETH) can be combined with bridgeMessage() 32
- I-04. yieldVault must be robust and safe 32
- I-05. Readability of expressions can be improved 33
- I-06. Different logic in _deposit() and completeMigration() 33
- I-07. Emits could be added 34
- I-08. Typos 35
- I-09. Gas optimizations 35

I-10. Missing error messages....36
I-11. Mistakes in fee calculator could cause a revert in completeMigration()....37
I-12. Potential rounding issues can be prevented....38
I-13. claimAndRedeem() could be frontrun with a call to claimAsset()....38
I-14. Some checks with force can be simplified....39
I-15. Unexpected Gas token can stay stuck in WETHNativeConverter....40
I-16. Function migrateGasBackingToLayerX() does not explicitly check sufficient gas tokens are available40
I-17. mint() function will not work if exchange ratio is ever changed due to share conversion rounding....40
Disclaimer....42
About Certora....42

certora polygon

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Vault bridge	https://github.com/agglayer/vault-bridge	Initial commit: 76081f6 Updated commit: 47f5a8c Last commit (after audit fixes): a40ed0f	EVM

Project Overview

This document describes the security assessment of **Vault bridge** using manual code review. The work was undertaken from **April 22 2025** to **May 6 2025.**

The following contract list is included in our scope:

```
src\CustomToken.sol
src\ITransferFeeCalculator.sol
src\VaultBridgeTokenInitializer.sol
src\VaultBridgeToken.sol
src\MigrationManager.sol
src\NativeConverter.sol
src\custom-tokens\GenericCustomToken.sol
src\custom-tokens\GenericNativeConverter.sol
src\custom-tokens\vbUSDC\VbUSDC.sol.generic
src\custom-tokens\vbUSDC\vbUSDCNativeConverter.sol.generic
src\custom-tokens\vbUSDS\VbUSDS.sol.generic
src\custom-tokens\vbUSDS\VbUSDSNativeConverter.sol.generic
src\custom-tokens\vbUSDT\VbUSDT.sol.generic
src\custom-tokens\vbUSDT\VbUSDTNativeConverter.sol.generic
src\custom-tokens\vbWBTC\VbWBTC.sol.generic
src\custom-tokens\vbWBTC\VbWBTCNativeConverter.sol.generic
```

```
src\custom-tokens\WETH\WETH.sol
src\custom-tokens\WETH\WETHNativeConverter.sol
src\etc\ERC20PermitUser.sol
src\etc\IBridgeMessageReceiver.sol
src\etc\ILxLyBridge.sol
src\etc\IUSDT.sol
src\etc\IVaultBridgeTokenInitializer.sol
src\etc\IVersioned.sol
src\etc\IWETH9.sol
src\vault-bridge-tokens\GenericVaultBridgeToken.sol
src\vault-bridge-tokens\vbETH\VbETH.sol
src\vault-bridge-tokens\vbUSDC\VbUSDC.sol.generic
src\vault-bridge-tokens\vbUSDS\VbUSDS.sol.generic
src\vault-bridge-tokens\vbUSDT\USDTTransferFeeCalculator.sol
src\vault-bridge-tokens\vbUSDT\VbUSDT.sol.generic
src\vault-bridge-tokens\vbWBTC\VbWBTC.sol.generic
```

The team performed a manual audit of all the Solidity contracts. During the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Protocol Overview

The Vault Bridge protocol is a yield-generating cross-chain bridge solution built on top of the LxLy Bridge system (Polygon zkEVM). Its core component, the Vault Bridge Token, combines ERC-4626 vault functionality with bridge mechanics to enable yield generation during asset bridging.

The protocol operates across two layers: Layer X (main network) hosts the Vault Bridge Token (ERC-20/4626) and a singleton Migration Manager for backing asset coordination, while Layer Y networks contain Custom Tokens (enhanced wrapped tokens) and Native Converters (pseudo-ERC-4626 vaults).

The tokens on Layer X will be held in a MetaMorpho 1.1 ERC-4626 vault.

The system supports bridging of major assets (WBTC, WETH, USDT, USDC, USDS) while producing yield, effectively solving the opportunity cost problem of traditional bridge locking periods through its vault-bridge hybrid architecture.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	2	2	2
Medium	3	3	3
Low	10	9	9
Informational	17	15	7
Total	32	29	21

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
H-01	Native tokens will be stuck in the bridge due to missing receive() function	High	Fixed
H-02	WETHNativeConverter. migrateGasBackingToLayerX() will always revert due to nonReentrant modifier in receive()	High	Fixed
M-01	Unexpected approval set due to wrong permit	Medium	Fixed
M-02	claimAndRedeem() will not work if exchange ratio is ever changed due to mixup with assets and shares	Medium	Fixed
M-03	Deposits may fail due to Morpho vault reporting inaccurate maxDeposit() values	Medium	Fixed
L-01	The allocation of underlying tokens between yieldVault and reserve can be manipulated	Low	Fixed
L-02	Fee tokens in combination with the yieldVault create a loss	Low	Fixed
L-03	Missing	Low	Fixed

	__ReentrancyGuard_init()		
L-04	yieldVault can be accessed even when paused, which can interfere with drainVault()	Low	Fixed
L-05	Denial of service for drainYieldVault() via donateAsYield()	Low	Not fixed
L-06	Function __Nonces_init() not called	Low	Fixed
L-07	Assets from migrateBackingToLayerX() could get stuck if changes are made in the bridge	Low	Fixed
L-08	Offline indexers could be confused by dual emits from _withdraw()	Low	Fixed
L-09	Unnecessary complexity in USDT transfer fee calculation logic	Low	Fixed
L-10	whenNotPaused modifier in admin-restricted functions may prevent recovery when contracts are paused	Low	Fixed

High Severity Issues

H-01 Native tokens will be stuck in the bridge due to missing receive() function

Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: WETHNativeConverter.sol MigrationManager.sol	Status: Fixed	

Description: The function `migrateGasBackingToLayerX()` of `WETHNativeConverter` sends native tokens to the `MigrationManager`. However, the `MigrationManager` does not have a `receive()` function so it cannot receive the gas token. As a consequence, it will not be possible to call `claimAsset()` on the receiving chain (Ethereum) and the tokens will remain stuck in the bridge.

```
JavaScript
function migrateGasBackingToLayerX(uint256 amount) ... {
    ...
    lxlyBridge().bridgeAsset{value: amount}(
        layerXLxlyId(), address(migrationManager()), amount, address(0), true, ""
    );
    ...
}
```

Recommendations: Add a `receive()` function to the `MigrationManager`. Alternatively, remove the call to `bridgeAsset()` in `migrateGasBackingToLayerX()` and send the tokens together with the `bridgeMessage()` call, as reported in I-03.

Customer's response: Fixed in commit c3b307f.

Fix Review: A `receive()` function was added.

H-02 WETHNativeConverter.migrateGasBackingToLayerX() will always revert due to nonReentrant modifier in receive()

Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: WETHNativeConverter.sol	Status: Fixed	

Description: The `nonReentrant` modifier on `WETHNativeConverter`'s `receive()` function prevents it from receiving ETH during `weth.bridgeBackingToLayerX()`, causing the transaction to revert. Because `migrateGasBackingToLayerX()` is also `nonReentrant`, the contract will be in an entered state.

```
JavaScript
receive() external payable whenNotPaused onlyIfGasTokenIsEth nonReentrant {}
```

Exploit Scenario: A migrator calls `migrateBackingToLayerX()`, which triggers the reentrancy guard. This function then calls `weth.bridgeBackingToLayerX()`, which in turn calls `WETHNativeConverter`'s `receive()` function, which checks the `ReentrancyGuard`'s `_status` flag. Because the flag is set, the call will revert, and so will the rest of the transaction.

Recommendations: Remove `nonReentrant` from `receive()` as it blocks legitimate ETH transfers.

Customer's response: Solved in commit a25c8d2 of PR 21.

Fix Review: Issue fixed.

Medium Severity Issues

M-01 Unexpected approval set due to wrong permit

Severity: **Medium**	Impact: **Low**	Likelihood: **High**
Files: NativeConverter.sol	Status: Fixed	

Description: The function `_deconvertWithPermit()` of `NativeConverter` applies a `permit` on the `underlyingToken`. However, it does not take in `underlyingToken` from the `msg.sender`, but `shares`. Note that since the `NativeConverter` already has the authority to burn shares, the use of a `permit` is not necessary.

```javascript
function _deconvertWithPermit(...) ... {
    ...
    _permit(address($.underlyingToken), assets, permitData);
    return _deconvert(shares, ...);
}
function _deconvert(...) ... {
    ...
    // Burn Custom Token.
    $.customToken.burn(msg.sender, shares);
    ...
}
```

Exploit Scenario: Call `deconvertWithPermit()`. This sets an approval for `underlyingToken`, which is not used. At a later moment, call `convert()`. Now, a larger amount of `underlyingToken` can be converted to `shares` than expected because a higher approval is set.
Note that this depends on the way the approval is made by the user: they could reset the previous approval, or increase the approval via `increaseAllowance()`.

However, the effect of this is limited because both actions are done by the owner of the `underlyingToken`.

Recommendations: Double check the usefulness of the functions `deconvertWithPermit()` and `deconvertWithPermitAndBridge()`, which call `_deconvertWithPermit()`, and consider removing them. Alternatively, change the code to use the `permit` of the `shares` and take the `owner` as an argument. In that case, also consider using `_spendAllowance()` .

Customer's response: Removed in commit `c3b307f`.

Fix Review: The functionality was removed entirely.

M-02 claimAndRedeem() will not work if exchange ratio is ever changed due to mixup with assets and shares

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: VaultBridgeToken.sol	Status: Fixed	

Description: The function `claimAndRedeem()` of `VaultBridgeToken` calls `_withdraw()` with a `shares` amount; however, this should be an `assets` amount.
For comparison, see the function `redeem()` which does use an `asset` amount.
As currently the `assets` and `shares` have a 1:1 conversion ratio, the code will work. However, if this changes in the future, the code will fail on the `require()` at the end of the function.

```
JavaScript
function claimAndRedeem(...) ... {
    ...
    assets = convertToAssets(amount);
    uint256 redeemedShares = _withdraw(amount, receiver, destinationAddress);
    require(redeemedShares == amount, ...);
}
function redeem(...) ... {
    assets = convertToAssets(shares);
    uint256 redeemedShares = _withdraw(assets, receiver, owner);
    require(redeemedShares == shares, ...);
}
```

Recommendations: Call `_withdraw()` with `assets` in argument.

Customer's response: Fixed in commit c3b307f.

Fix Review: The call was updated to pass the correct argument.

M-03 Deposits may fail due to Morpho vault reporting inaccurate maxDeposit() values

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: VaultBridgeToken.sol	Status: Fixed	

Description: The `_depositIntoYieldVault()` function of `VaultBridgeToken` relies on the return value of `yieldVault.maxDeposit()` to determine how many assets can be deposited into the yield vault:

```
JavaScript
    uint256 maxDeposit_ = $.yieldVault.maxDeposit(address(this));
    ...
    // Set the return value.
    nonDepositedAssets = assets > maxDeposit_ ? assets - maxDeposit_ : 0;

    // Calculate the amount to deposit into the yield vault.
    assets = assets > maxDeposit_ ? maxDeposit_ : assets;
    ...
    uint256 mintedYieldVaultShares = $.yieldVault.deposit(assets, address(this));
```

However, when using MetaMorpho as the yield vault, its `maxDeposit()` function explicitly warns:

```
JavaScript
    /// @dev Warning: May be higher than the actual max deposit due to duplicate markets in
the supplyQueue.
    function maxDeposit(address) public view override returns (uint256) {
```

If the reported limit is inaccurate, the subsequent deposit call will revert, blocking all deposits that attempt to forward more than the actual `maxDeposit` amount into the yield vault.

Recommendations: Wrap the deposit call in a try/catch block to gracefully handle failures. In the catch block, update `nonDepositedAssets` to include the assets that couldn't be deposited.

Customer's response: Morpho responded that "there is no reason why a curator would do this". Nevertheless, a `try/catch` was added in PR 21.

Fix Review: Issue fixed.

Low Severity Issues

L-01 The allocation of underlying tokens between yieldVault and reserve can be manipulated

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: VaultBridgeToken.sol#L366-L445 VaultBridgeToken.sol#L591-L661	Fixed	

Description: The functions `_deposit()` and `withdraw()` of `VaultBridgeToken` are asymmetrical. This allows users to shift tokens from the `reserve` to the `yieldVault` and the other way around. If this isn't detected and corrected quickly via `_rebalanceReserve()` , yield could be lost or withdrawals could be affected.
Also in case of tokens that have a fee, losses could be created by large movements in and out of the `yieldVault`.

Scenario 1: Convert `yieldVault` holdings to reserves

1) Get a flashloan of `VaultBridgeToken` shares, larger than the reserve and the `minimumYieldVaultDeposit`;
2) Do a large `withdraw()` that also does a large `withdraw()` from the `yieldVault`;
3) Repeatedly `deposit()` an amount `< minimumYieldVaultDeposit`. This is added to the `reserves` and not put in the `yieldVault`;
4) Pay off flashloan.

Now a (large) amount is shifted from the `yieldVault` to the `reserves` and no yield is being earned.

Scenario 2: Convert `reserves` to `yieldVault`

1) Obtain a flashloan of assets;
2) Make a large `deposit()`;
3) Perform a large `withdraw()`, which depletes the reserve.

If the reserve is depleted, all withdrawals must go through the yield vault. If the yield vault is paused or blocked, users will be unable to withdraw from the bridge token vault, resulting in a potential DoS. This also removes any safety buffer the reserve was meant to provide, exposing all funds to risks in the yield vault.

```JavaScript
function _deposit(...) ... {
    ...
    // Check whether to skip depositing into the yield vault.
    if (assets >= $.minimumYieldVaultDeposit) {
        ...
        ..._depositIntoYieldVault(assetsToDeposit, false);
        ...
    }
}
function _withdraw(...) ... {
    ...
    if (maxWithdraw_ >= remainingAssets) {
        ...
        _withdrawFromYieldVault(...);
        ...
    }
}
```

Recommendations: Consider adjusting the reserve comparable to `_rebalanceReserve()` in both `_deposit()` and `_withdraw()`.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

L-02 Fees tokens in combination with the yieldVault create a loss

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: VaultBridgeToken.sol	Fixed	

Description: In the case of fee-on-transfer tokens, the `VaultBridgeToken` must pay transfer fees when depositing assets into the `yieldVault`. Currently, this will induce a loss for the protocol. As shown by the below code comment in the contract, this is a known risk:

```
Unset
It is expected that generated yield will offset any costs incurred when transferring the underlying token to and from the yield vault, or depositing to and withdrawing from the yield vault for the purpose of generating yield or rebalancing the internal reserve.
```

Nevertheless, it is important to note that:

- This may have a non-trivial impact in combination with **L-01 The allocation of underlying token between yieldVault and reserve can be manipulated**.
- It can be purposefully exploited to DoS `VaultBridgeToken` withdrawals, albeit at a cost at least twice as large as the induced loss. When a loss as large as any accumulated yield + 1% of the supply has been caused, the check in `_withdrawFromYieldVault()` will block all withdrawals once the reserve has been depleted.

Recommendations: Double check if the loss should not be passed on to the user.

Customer's response: Tokens with a fee-on-transfer are no longer supported. Solved in PR 21.

Fix Review: Issue fixed.

L-03 Missing __ReentrancyGuard_init()

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: MigrationManager.sol CustomToken.sol	Fixed	

Description: The contracts `CustomToken` and `MigrationManager` inherit from `ReentrancyGuardUpgradeable`, but their `initialize()` function doesn't call `__ReentrancyGuard_init()`.
For comparison they do call other `__*_init()` functions. The risk is limited because the initializer is empty.

```
JavaScript
import {ReentrancyGuardUpgradeable} from
"@openzeppelin-contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
```

Recommendations: Consider also calling `__ReentrancyGuard_init()`, in both `CustomToken` and `MigrationManager`.

Customer's response: Fixed in commit b619367.

Fix Review: Issue fixed.

L-04 yieldVault can be accessed even when paused, which can interfere with drainVault()

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: VaultBridgeToken.sol#L933 VaultBridgeToken.sol#L955 VaultBridgeToken.sol#L1137	Fixed	

Description: Several functions in `VaultBridgeToken` do not use the `whenNotPaused` modifier. While some functions (such as `drainVault()` and `donateAsYield()`) are safe and may need to remain callable when the contract is paused, other functions can still interact with the `yieldVault` even when paused, which may be undesirable.

While a comment in `drainVault()` suggests that deposits should be stopped (by setting `minimumYieldVaultDeposit` to `uint256.MAX`), it is also important to ensure that functions like `_rebalanceReserve()` cannot be called while paused, as they could still interact with the `yieldVault`. Specifically, the `burn()` function can call `_rebalanceReserve()`.

JavaScript

```
  /// @notice Make sure to disable deposits by setting minimumYieldVaultDeposit to
uint256.MAX
function drainVault(uint256 amountToDrain_, bool exact) external onlyRole(DEFAULT_ADMIN_ROLE)
nonReentrant { ... }
function donateAsYield(uint256 assets) external nonReentrant { ... }
function burn(uint256 shares) external onlyYieldRecipient nonReentrant { ... }
    ...
    _rebalanceReserve(false, true);
    ...
}
```

Exploit Scenario: A problem is detected with the `yieldVault`. Someone with `PAUSER_ROLE` pauses the `VaultBridgeToken` contract. An admin calls `drainYieldVault()`. Other entities that have noticed the problem and call `donateAsYield()` to help fix the issue.

The `YieldRecipient()` calls `burn()` to help fix the issue. Via `burn()`, the function `_rebalanceReserve()` is called and the `yieldVault` is filled again.

Recommendations: Document that `pause()` should be used when using `drainVault()`. Consider changing the `burn()` function in order to prevent triggering a `_rebalanceReserve()` when the contract is paused.

Customer's response: Solved in commit 4abf293 of PR 21.

Fix Review: Issue fixed.

L-05 Denial of service for drainYieldVault() via donateAsYield()

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: VaultBridgeToken.sol	Not Fixed	

Description: The function `drainYieldVault()` of `VaultBridgeToken` contains a calculation where `originalReservedAssets` is subtracted from `convertToAssets (originalTotalSupply)`.

However, `originalReservedAssets` can be arbitrarily increased by anyone via `donateAsYield()`, which could lead to a revert and thus a denial of service on `drainYieldVault()`.
Note: the attack could be costly depending on the state of the `yieldVault`.

The comparable function `_withdrawFromYieldVault()` has a similar calculation but includes `originalUncollectedYield`, which prevents the revert.

```JavaScript
function drainYieldVault(...) ... {
    ...
    uint256 originalReservedAssets = $.reservedAssets;
    ...
    require(Math.mulDiv(..., ..., ...) >=
        Math.mulDiv(convertToAssets(originalTotalSupply) - originalReservedAssets, ..., ...)
    );
    ...
}
function donateAsYield(...) ...
    ...
    assets = _receiveUnderlyingToken_assets(msg.sender, assets);
    $.reservedAssets += assets;
}
function _withdrawFromYieldVault(
    ...
    require(Math.mulDiv(convertToAssets(originaltotalSupply + originalUncollectedYield)
```

```
                        - originalReservedAssets, ..., ...)) <= ...)
    ...
}
```

Exploit Scenario: There is an issue with the `yieldVault` and the admin has paused the contracts. Someone tries to fix the issue by calling `donateAsYield()` with a large amount of assets.
The admin tries to call `drainYieldVault()`, but this reverts because the amount of `reservedAssets` is larger than `totalSupply()`.

Recommendations: Incorporate the yield calculation in `drainYieldVault()`.

Customer's response: That was done to relax the check so the vault can be drained even if the act consumes some yield. What can be done instead is to collect yield before calling the function. If too much gets collected for some reason, it is always possible to donate it back.
We can add this to the docs.

Fix Review: The manual workaround will sufficiently fix this.

L-06 Function __Nonces_init() not called

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: VaultBridgeTokenInitializer.sol VaultBridgeToken.sol CustomToken.sol ERC20PermitUpgradeable.sol NoncesUpgradeable.sol	Fixed	

Description: The `initialize()` function of `VaultBridgeTokenInitializer` calls `__ERC20Permit_init()`, however it doesn't call the function `__Nonces_init()` which is a part of `ERC20PermitUpgradeable`.
The same issue is present in `__CustomToken_init()` of `CustomToken`.
The risk is limited because the function `__Nonces_init()` is currently empty, but this could change in the future.

```JavaScript
// VaultBridgeTokenInitializer.sol
contract VaultBridgeTokenInitializer is ... , VaultBridgeToken {
    function initialize(...) {
        ...
        __ERC20Permit_init(initParams.name);
        ...
    }
}
// VaultBridgeToken
import {ERC20PermitUpgradeable} from
    "@openzeppelin-contracts-upgradeable/token/ERC20/extensions/ERC20PermitUpgradeable.sol";

// ERC20PermitUpgradeable
import {NoncesUpgradeable} from "../../../utils/NoncesUpgradeable.sol";
abstract contract ERC20PermitUpgradeable is ..., NoncesUpgradeable { ... }
```

```
// NoncesUpgradeable.sol
function __Nonces_init() internal onlyInitializing { }
```

Recommendations: Consider also calling function `__Nonces_init()` in both `CustomToken` and `VaultBridgeTokenInitializer`.
According to OpenZeppelin, `__ERC20Permit_init()` doesn't call `__Nonces_init()` because it is intended to replicate the behavior of an abstract contract and is done to avoid duplicated initializers if the ERC20Permit is combined with other contracts inheriting from Nonces.
Note: the risk is limited because the function `__Nonces_init()` is currently empty.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

L-07 Assets from migrateBackingToLayerX() could get stuck if changes are made in the bridge

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: NativeConverter.sol VaultBridgeToken.sol	Fixed	

Description: The function `migrateBackingToLayerX()` of `NativeConverter` checks the balances of the `bridge` to detect if any transfer fees have been deducted. However, this check makes use of implementation details of the bridge. If the bridge code ever changes, for example by transferring the tokens to a different contract, then this logic will no longer work.
In that case `assets` could be `0` and a misleading message would be sent with `bridgeMessage()`. The assets will be bridged to the `MigrationManager` where they will be stuck because `assets==0`. Then the `assets` from the `assetsInMigrationFund` will be used if available.

```
JavaScript
uint256 balanceBefore = $.underlyingToken.balanceOf(address($.lxlyBridge));
$.lxlyBridge.bridgeAsset(..., assets, address($.underlyingToken), ...);
assets = $.underlyingToken.balanceOf(address($.lxlyBridge)) - balanceBefore;
```

Recommendations: In `migrateBackingToLayerX()`, check the amount of sent `assets` is plausible (e.g. after the `balanceOf(bridge)` check).
Also consider checking that `convertToAssets(shares)` is approximately equal to `assets` in function `completeMigration()`. Alternatively, calculate the transfer fee or change the bridge to return the sent amount, which could be useful for other projects too.

Customer's response: Solved in PR 21 by supporting only underlying tokens which the bridge can mint/burn natively. Even if such a token has a transfer fee, the bridge burns the token directly (without transferring it when bridging), so the transfer fee does not apply.

Fix Review: Issue fixed.

L-08 Offline indexers could be confused by dual emits from _withdraw()

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: VaultBridgeToken.sol	Fixed	

Description: Function `_withdraw()` of `VaultBridgeToken` potentially does `_burn()` and `transfer()` twice. This could be misleading for the `receiver`.
Both `_burn()` and `transfer()` do an `emit` which is used by offline indexers. These offline indexers could be confused by the dual emits.

```
JavaScript
function _withdraw(...) ... {
    ...
    _burn(owner, convertToShares(amountToWithdraw)); // first burn
    ...
    _sendUnderlyingToken(receiver, amountToWithdraw); // first transfer()
    ...
    _burn(owner, convertToShares(remainingAssets)); // second burn
    ...
    _withdrawFromYieldVault(..., receiver, ... ); // second transfer()
    ...
}
```

Recommendations: Consider combining the two `_burn()` actions into one.
Also consider combining the `transfer()` actions. This can be done by first exercising the `_withdrawFromYieldVault()` to the `VaultBridgeToken` contract and then a `transfer()` of the full amount to the `receiver`. Note: this would lead to additional fees for fee-on-transfer tokens.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

L-09 Unnecessary complexity in USDT transfer fee calculation logic

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: USDTTransferFeeCalculator.sol	Fixed	

Description: The calculation in `assetsBeforeTransferFee()` is unnecessarily complex. The function contains a superfluous while-loop and `maximumFee` check:

```
JavaScript
        while (candidate > 0) {
            uint256 feeCandidateMinus1 = ((candidate - 1) * basisPointsRate) / 10000;
            if (feeCandidateMinus1 > maximumFee) { // <- we know c-1 is below max-fee
                feeCandidateMinus1 = maximumFee;   // threshold, because c is
            }

            uint256 afterFeeMinus1 = (candidate - 1) - feeCandidateMinus1;
            if (afterFeeMinus1 >= minimumAssetsAfterTransferFee_) {
                candidate--; // <- the reverse calculation will at most be off by one
            } else {
                break;
            }
        }
```

This introduces unnecessary complexity and requires slightly more gas.

Recommendations: The above snippet can be simplified to the below logic without loss of generality:

```
JavaScript
    uint256 candidateMinusOne = candidate - 1;
    uint256 feeForCMO = (candidateMinusOne * basisPointsRate) / 10000;
    uint256 CmoMinusFee = candidateMinusOne - feeForCMO;
```

```
return (CmoMinusFee == minimumAssetsAfterTransferFee_) ? candidateMinusOne : candidate;
```

The equivalence of the two versions was verified via a non-exhaustive fuzz test.

Customer's response: Tokens with a fee-on-transfer are no longer supported. Solved in PR 21.

Fix Review: Issue fixed.

L-10 whenNotPaused modifier in admin-restricted functions may prevent recovery when contracts are paused

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: MigrationManager.sol NativeConverter.sol	Fixed	

Description: The functions `configureNativeConverters()` of `MigrationManager` and `setNonMigratableBackingPercentage()` of `NativeConverter` could be valuable for recovery operations during paused states. However, both include the `whenNotPaused` modifier despite already being restricted to addresses with `DEFAULT_ADMIN_ROLE`. This is redundant as admins can unpause, execute the function, and pause again if necessary.

Exploit Scenario: During an emergency that requires the contracts to remain paused, admins need to reconfigure token mappings or adjust backing percentages. The current implementation forces an unnecessary unpause-execute-pause cycle.

Recommendations: Reconsider if the `whenNotPaused` modifier is strictly necessary for these functions, or if keeping them accessible during paused states could be more practical.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

Informational Issues

I-01. Unused struct NativeConverterConfiguration

Description: Contract `VaultBridgeToken` defines `struct NativeConverterConfiguration` but this is never used.

Recommendation: Consider removing `struct NativeConverterConfiguration`.

Customer's response: Fixed in commit b619367.

Fix Review: Verified

I-02. Reentrancy library could also use transient storage

Description: The library `ReentrancyGuardUpgradeable` is used in several locations. There is also a transient storage version which uses less gas.

Recommendation: Consider using ReentrancyGuardTransientUpgradeable.sol.

Customer's response: Implemented in the contracts that are deployed on mainnet in PR 21.

Fix Review: Issue fixed.

I-03. bridgeAsset() of native asset (ETH) can be combined with bridgeMessage()

Description: The function `migrateGasBackingToLayerX` of `WETHNativeConverter` calls both `bridgeAsset()` with a native asset (ETH) amount and `bridgeMessage()`. However, these could also be combined, which simplifies the bridge actions.

Recommendation: Consider sending the native asset (ETH) with `bridgeMessage()`.

Customer's response: Acknowledged.

Fix Review: Acknowledged.

I-04. yieldVault must be robust and safe

Description: It is important that the `yieldVault` is robust and safe. To aid with verifying this we have summarized the following attention points.

Recommendation: Check the `yieldVault` has taken the following into account:
- all management actions on the `yieldVault`, including upgrades are well secured;
- the probability of value loss is very low;
- inflation attack is taken into account;
- the ratio between `assets` and `shares` cannot be seriously shifted with sandwich attacks;
- the amount of `assets` returned by `withdraw()` is always equal to the amount requested;
- the amount of `shares` reported by `deposit()` is always to the amount of shares minted;
- the amount of assets / shares transferred by `withdraw()`, `redeem()`, `deposit()` is equal to the amount specified;
- the functions `maxWithdraw()`, `maxDeposit()`, `convertToAssets()`, `balanceOf()`, `maxRedeem()` give reliable information;
- the `yieldVault` only transfers tokens from the `VaultBridgeToken` when explicitly requested to do so via `deposit()`.

Customer's response: Forwarded to DeFi team.

Fix Review: Acknowledged.

I-05. Readability of expressions can be improved

Description: The code contains some frequently used expressions that could be made more readable with a function.

Recommendation: Create functions for the following expressions, or use a library:

- `a > b ? a : b` // max()
- `a < b ? a : b` // min()
- `a > b ? a-b : 0` // saturating subtract

Customer's response: Acknowledged.

Fix Review: Acknowledged.

I-06. Different logic in _deposit() and completeMigration()

Description: The function `_deposit()` of VaultBridgeToken.sol#L356-L434 and `completeMigration()` of VaultBridgeToken.sol#L972-L1036 are more or less the same, as they both move tokens to the `yieldVault`.
However, the implementation is different: `_deposit()` also checks `$.minimumYieldVaultDeposit`, while `completeMigration()` doesn't. This could have unexpected results and makes the code more difficult to maintain.

```JavaScript
function _deposit(...) ... {
    ...
    uint256 assetsToReserve = _calculateAmountToReserve(assets, shares);
    uint256 assetsToDeposit = assets - assetsToReserve;
    ...
    if (assetsToDeposit >= $.minimumYieldVaultDeposit) {
        ... _depositIntoYieldVault(assetsToDeposit, false);
    }
    ...
}
function completeMigration(...) ... {
```

```
    ...
    uint256 assetsToReserve = _calculateAmountToReserve(assets, shares);
    uint256 assetsToDeposit = assets - assetsToReserve;
    ...
    ... _depositIntoYieldVault(assetsToDeposit, false);
}
```

Recommendation: Consider making the logic in both functions equal. Preferably move duplicate logic to a function, for example by doing the check for `$.minimumYieldVaultDeposit` inside of `calculateAmountToReserve()`, which seems like a logical location.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

I-07. Emits could be added

Description: In some places more `emits` or `emit` parameters could be added.

Recommendation: Consider adding the following:

- MigrationManager.sol#L211 : `underlyingToken` could be added;
- VaultBridgeToken.sol#L1184-L1192 : an `emit MinimumDepositAmountSet` could be added.

Customer's response: Acknowledged

Fix Review: Acknowledged.

I-08. Typos

Description: Some typos are present in the code:

- CustomToken.sol#L18 : `IMPROTANT` should be `IMPORTANT`

Recommendation: Consider fixing the typos.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

I-09. Gas optimizations

Description: The code can be optimized in the following places to save gas.

Recommendation:

- MigrationManager.sol#L173 : `networkID()` is used inside a loop, could be cached before the loop;
- MigrationManager.sol#L202-L204 : `oldTokens` could be used;
- MigrationManager.sol#L188-L190 and MigrationManager.sol#L202-L204 : the two instances of `forceApprove()` could be combined and moved before the `if` of MigrationManager.sol#L180;
- VaultBridgeToken.sol#L417 : `$.reservedAssets` could be replaced with `assetsToReserve`;
- VaultBridgeToken.sol#L615 : `$.reservedAssets` could be replaced with `originalReservedAssets`;
- VaultBridgeToken.sol#L854 : `totalSupply()` could be replaced with `originalTotalSupply`;
- VaultBridgeToken.sol#L1009-L1037 : the updates of `$.reservedAssets` can be combined in the following way:

```
JavaScript
uint256 requiredAssets = convertToAssets(shares);
uint256 discrepancy = requiredAssets - assets;
...
$.migrationFeesFund_assets -= discrepancy;
```

```
$.reservedAssets += discrepancy; // requiredAssets includes the discrepancy
uint256 assetsToReserve = _calculateAmountToReserve(assets requiredAssets, shares);
uint256 assetsToDeposit = assets requiredAssets - assetsToReserve;
if (assetsToDeposit > 0) {
    $.reservedAssets assetsToReserve += _depositIntoYieldVault(assetsToDeposit, false);
}
$.reservedAssets += assetsToReserve;
```

Customer's response: Solved in PR 21.

Fix Review: Verified: most suggestions have been implemented.

I-10. Missing error messages

Description: In some situations, no custom error message is given.

Recommendation: Consider adding a custom error message:

- VaultBridgeToken.sol#L1163-L1166 : an error message like `InsufficientAssetsReceived` could be added.

Customer's response: Solved in PR 21.

Fix Review: Issue fixed.

I-11. Mistakes in fee calculator could cause a revert in completeMigration()

Description: It is important that in the fee calculator: USDTTransferFeeCalculator.sol the result of `assetsAfterTransferFee()` is never higher than the input.
This is because `completeMigration()` of VaultBridgeToken.sol#L984-L1048 receives values for `shares` and `assets` that originate from `migrateBackingToLayerX()` of NativeConverter.sol#L481-L525. The value of `assets` is converted via the function `_assetsAfterTransferFee()`. If the fee calculations were rounded up, then slightly more assets would be received than sent. If this occurred, the calculation of `discrepancy` in `completeMigration()` would revert.

```JavaScript
function migrateBackingToLayerX(uint256 assets) ... {
    ...
    uint256 shares = _convertToShares(assets);
    ...
    $.lxlyBridge.bridgeMessage(..., abi.encode(... , abi.encode(shares, assets)));
}
function completeMigration(uint32 originNetwork, uint256 shares, uint256 assets)
    ...
    assets = _assetsAfterTransferFee(assets); // assets may not increase
    ...
    uint256 requiredAssets = convertToAssets(shares);
    uint256 discrepancy = requiredAssets - assets; // potential revert in edge case
    ...
}
```

Recommendations: Consider documenting that the result of `assetsAfterTransferFee()` must be always lower or equal to the input.
Alternatively the code of `completeMigration()` could be made more robust.

Customer's response: Fee-on-transfer tokens are no longer supported, solved in PR 21.

Fix Review: Issue fixed.

I-12. Potential rounding issues can be prevented

Description: There are a few calculations of the type `a * b / c <= d`. This could potentially lead to rounding issues.

Recommendation: Consider changing the code to use the following equivalent expression: `a * b <= c * d`, which removes the division and thus the potential rounding issues.

Customer's response: The impact of the potential rounding issues is very small. Unchanged for readability.

Fix Review: Acknowledged.

I-13. claimAndRedeem() could be frontrun with a call to claimAsset()

Description: Function `claimAndRedeem()` of VaultBridgeToken.sol#L699-L736 calls `claimAsset()` to claim the funds from the bridge. However, `claimAsset()` is permissionless and other parties may have called it before (frontrun). In that case, `claimAndRedeem()` will revert and a separate `withdraw()` has to be done. This might not be obvious to users of `claimAndRedeem()`.

Recommendation: Document this scenario.

Customer's response: The documentation will be updated.

Fix Review: Acknowledged.

I-14. Some checks with force can be simplified

Description: In functions `_simulateWithdraw()` of VaultBridgeToken.sol#L541-L566 and `_simulateDeconvert()` of NativeConverter.sol#L295-L320, the extra requirement after the `if (force)` checks isn't necessary, because it is already enforced before.

```
JavaScript
function _simulateWithdraw(...) ... {
    ...
    if (remainingAssets == maxWithdraw_) return assets;
        remainingAssets -= maxWithdraw_;
        // now remainingAssets > 0, otherwise it would have returned above
        ...
        if (force) require(remainingAssets == 0, AssetsTooLarge(withdrawnAssets, assets));
    }
}
```

```
JavaScript
function _simulateDeconvert(...) ... {
    ...
    if (backingOnLayerY_ >= remainingAssets) return shares;
    remainingAssets -= backingOnLayerY_;
    // now remainingAssets > 0 otherwise it would have returned above
    ...
    if (force) require(remainingAssets == 0, AssetsTooLarge(convertedAssets, assets));
}
```

Recommendation: In both functions consider replacing `require(remainingAssets == 0,...)` with `revert(...)`.

Customer's response: Those checks are included so that additional functionality can be added above the last line, without requiring refactoring of previous code.

Fix Review: Acknowledged.

I-15. Unexpected Gas token can stay stuck in WETHNativeConverter

Description: The function `receive()` of WETHNativeConverter.sol#L161 can receive gas tokens from any contract. However, they will only be processed if they are received from WETH.sol during a call from `migrateGasBackingToLayerX()`. Otherwise, they will stay stuck in the contract.

Recommendation: Consider checking in `receive()` that the `msg.sender`is `WETH.sol`.

Customer's response: Suggestion not implemented.

Fix Review: Acknowledged.

I-16. Function migrateGasBackingToLayerX() does not explicitly check sufficient gas tokens are available

Description: Function `migrateGasBackingToLayerX()` of WETHNativeConverter.sol#L119-L159 doesn't explicitly check enough gas tokens have been received from WETH.sol.
The advantage of an explicit check is that a relevant custom error can be given, which makes troubleshooting configuration errors easier.

Recommendation: Consider checking sufficient funds before the call to `bridgeAsset()`.

Customer's response: Suggestion not implemented.

Fix Review: Acknowledged.

I-17. mint() function will not work if exchange ratio is ever changed due to share conversion rounding

Description: Function `_deposit()` of VaultBridgeToken.sol converts `maxShares` to `assets` and then back to `shares`. This currently has no impact since assets and shares have a 1:1 conversion ratio, but if this changes and rounding is applied, then the `require()` at the end of `mint()` will cause it to revert.

```JavaScript
function mint(uint256 shares, ...) ... {
    (mintedShares, assets) =
        _deposit(_assetsBeforeTransferFee(convertToAssets(shares)), ... , shares);
    require(mintedShares == shares, ...);
}
function _deposit(uint256 assets, ..., uint256 maxShares) ... returns (uint256 shares, ...) {
    ...
      // Calculate the required amount of the underlying token.
      uint256 requiredAssets = convertToAssets(maxShares); // first conversion
      ...
             assets = requiredAssets;
    ...
    shares = convertToShares(assets); // second conversion
    ...
    spentAssets = assets;
    ...
    _mint(... , shares);
    ...
}
```

Recommendations: Convert `assets` to shares and compare the resulting `shares` to `maxShares`, rather than converting `maxShares` to assets. This will ensure the comparison happens in the target unit (shares) rather than through an intermediate conversion.

Customer's response: Suggestion not implemented. In the convert function there is a comment:

```Unset
CAUTION! Changing this function will affect the conversion rate for the entire contract, and
may introduce bugs.
```

As it is unlikely the ratio will ever change, any adaptation of the convert functions should be left for whoever is doing it in the future.

Fix Review: Acknowledged.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.