certora

Security Assessment
Final Report

polygon

Vault Bridge

June 2025

Prepared for Polygon Labs

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Plat-form
Vault bridge	https://github.com/agglayer/vault-bridge	Initial commit: 3a7d025 Fix review commit: e4243e0	EVM

Project Overview

This document describes the security assessment of **Vault bridge** using formal verification. The work was undertaken from **May 13 2025** to **June 23 2025.**

The following contract list is included in our scope:

```
src\CustomToken.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\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\VbUSDT.sol.generic
src\vault-bridge-tokens\vbWBTC\VbWBTC.sol.generic
```

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. During the verification process, the Certora team identified formal specifications and proofs, as listed on the following pages.

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.

Detailed Findings

ID	Title	Severity	Status
I-01	Incorrect rounding in the calculation of `minimumReserve`	Informational	Fixed
I-02	Incorrect rounding in the calculation of `nonMigratableBacking`	Informational	Fixed

Informational Severity Issues

I-01. Incorrect rounding in the calculation of `minimumReserve`

Description: In the `_rebalanceReserve()` function in `VaultBridgeToken.sol` we have the following line:

```
None
uint256 minimumReserve = convertToAssets(Math.mulDiv(originalTotalSupply,
$.minimumReservePercentage, 1e18));
```

By default the `mulDiv` rounds down so the `minimumReserve` could end up being off by one. Consequently, the actual percentage of reserved assets can drop below the `minimumReservePercentage`, as demonstrated by the violated rule in P-03.

Example:

```
    totalSupply() = 10
    minimumReservePercentage = 39 % = 39e16
    minimumReserve = convertToAssets(Math.mulDiv(10, 39e16, 1e18) = 3
    Actual reserved percentage = 3 / 10 = 30 % < 39 %
```

The same formula is used also in the function `_calculateAmountToReserve()`.

Recommendation: Specify the rounding direction manually.

Customer's response: Fixed

I-02. Incorrect rounding in the calculation of `nonMigratableBacking`

Description: In the `migratableBacking()` function in `NativeConverter.sol` the `nonMigratableBacking` is computed as follows:

None

```
uint256 nonMigratableBacking =
_convertToAssets(Math.mulDiv(customToken().totalSupply(),
$.nonMigratableBackingPercentage, 1e18));
```

By default the `mulDiv` rounds down so the result could end up being off by one. Consequently, the actual percentage of backing can drop below the `nonMigratableBackingPercentage`. This is shown in the violated rule in P-05.

Recommendation: Specify the rounding direction manually.

Customer's response: Fixed

Formal Verification

Verification Methodology

We performed verification of the **Polygon** protocol using the Certora verification tool which is based on Satisfiability Modulo Theories (SMT). In short, the Certora verification tool works by compiling formal specifications written in the Certora Verification Language (CVL) and **Polygon**'s implementation source code written in Solidity.
More information about Certora's tooling can be found in the Certora Technology Whitepaper.

If a property is verified with this methodology it means the specification in CVL holds for all possible inputs. However specifications must introduce assumptions to rule out situations which are impossible in realistic scenarios (e.g. to specify the valid range for an input parameter). Additionally, SMT-based verification is notoriously computationally difficult. As a result, we introduce overapproximations (replacing real computations with broader ranges of values) and underapproximations (replacing real computations with fewer values) to make verification feasible.

Rules: A rule is a verification task possibly containing assumptions, calls to the relevant functionality that is symbolically executed and assertions that are verified on any resulting states from the computation.

Inductive Invariants: Inductive invariants are proved by induction on the structure of a smart contract. We use constructors as a base case, and consider all other (relevant) externally callable functions that can change the storage as step cases.
Specifically, to prove the base case, we show that a property holds in any resulting state after a symbolic call to the respective constructor. For proving step cases, we generally assume a state where the invariant holds (induction hypothesis), symbolically execute the functionality under investigation, and prove that after this computation any resulting state satisfies the invariant.

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

- We use mock contracts for `asset() / underlyingToken, lxlyBridge` and `yieldVault`. These exhibit the typical behavior of the respective contracts. By doing this we implicitly assume that `underlyingToken != GenericVaultBridgeToken`.

- We assume that `underlyingToken`, `lxlyBridge` and `yieldVault` work correctly and do not have any security vulnerabilities that an attacker could exploit, that is, we assume that `yieldVault` is immune to inflation attacks.

- We use our own implementation of `Math.mulDiv` which is equivalent to the original and better suited to our prover.

- Loops are inherently difficult for formal verification. We handle loops by unrolling them a specific amount of times. We use a **loop_iter of 2**, unrolling each loop exactly twice. This only affects the loop in `MigrationManager.configureNativeConverters`.

- Harnessing: We work with contracts inherited from the original contracts to add additional methods, flags, getters, etc. for verification purposes without modifying the original code. Any verification result on the harness applies to the original contract.

- The verified Solidity contracts are compiled with solcX (with via-ir).

Formal Verification Properties Overview

ID	Contract	Title	Impact	Status
P-01	GenericVault BridgeToken	Solvency	Verifies that the vault token maintains solvency against its total obligations under worst-case yield slippage.	Verified
P-02	GenericVault BridgeToken	Valid States Invariants	Prevents invalid configurations by enforcing bounds on reserves, supply, and allowances.	Verified
P-03	GenericVault BridgeToken	Integrity Rules	Verifies correctness and monotonicity of core financial operations to ensure reliable and predictable behavior.	Verified after fix
P-04	GenericVault BridgeToken	Risk assessment	Guarantees that only known and permitted methods can mutate critical state variables preventing unauthorized behavior.	Verified
P-05	GenericNativeConverter MigrationManager	Risk assessment	Verifies that the system remains solvent and migration-safe, with sufficient and non-depletable asset backing.	Verified after fix

Detailed Properties

GenericVaultBridgeToken

Module Properties

To check correctness of the GenericVaultBridgeToken, we prove **four sets of properties**:

P1 **Solvency.** Let's use this notation:

A = `convertToAssets(totalSupply() + yield()) - reservedAssets()`
B = return value of `yieldVault.withdraw(assets)`
C = `assets`
D = `yieldVault.balanceOf(GenericVaultBridgeToken)`
E = `(10^18 + yieldVaultMaximumSlippagePercentage)`

The solvency property states that for all `assets`

```
A * B / C <= D * E / 10^18
```

This can be rewritten to

```
A * B * 10^18 <= D * E * C
```

We're assuming that the `yieldVault` always holds more assets than its total shares which means that `B <= C`, so

```
A * B * 10^18 <= A * C * 10^18 (<= D * E * C)
```

Hence we can cancel the term `C` and simplify the formula to

```
A * 10^18 <= D * E
```

We formally prove this as rule `vaultBridgeTokenSolvency`.

We also proved a simplified version which states

```
totalAssets() >= convertToAssets(totalSupply())
```

This is verified as rule `vaultBridgeTokenSolvency_simple`.

P2 **Valid states of the contract:** a set of invariants that hold in every valid state. E.g.

`minimumReservePercentage <= 10^18`.

We proved these and then we assume them in our other rules.

P3 **Integrity of important methods**

P4 **Risk assessment properties**

P-01. Solvency

Status: Verified

Rule Name	Status	Description	Link to rule report
vaultBridgeTokenSolvency_simple **vaultBridgeTokenSolvency**	Verified	*The contract stays solvent.*	[Report]()

P-02. Valid states

Status: Verified

Rule Name	Status	Description	Link to rule report
minimumReservePercentageLimit	Verified	`minimumReservePercentage` *cannot exceed* `10^18.`	[Report]()
netCollectedYieldAccounted **netCollectedYieldLimited**	Verified	`netCollectedYield` *is never greater than the balance of the* `yieldRecipient.`	[Report]()
reserveBacked	Verified	*The contract holds enough underlying tokens to cover* `reservedAssets + migrationFeesFund.`	[Report]()
assetsMoreThanSupply **noSupplyIfNoAssets**	Verified	`totalSupply` *cannot exceed* `totalAssets.`	[Report]()
zeroAllowanceOnAssets **zeroAllowanceOnShares**	Verified	*The contract doesn't give allowance to any address except the* `yieldVault.`	[Report]()

P-03. Integrity of important methods

Status: Verified after fix

Rule Name	Status	Description	Link to rule report
integrityOfRebalance	Verified after fix Reported issue I-01	*After calling* `rebalanceReserve`*, the* `reservedAssets` *>=* `minimumReservedAssets`*. The method doesn't affect* `totalAssets`*.*	Original report Report after fix
integrityOfRebalance_margin1	Verified	*After calling* `rebalanceReserve`*, the* `reservedAssets` *+1 >=* `minimumReservedAssets`*. The method doesn't affect* `totalAssets`*.*	Report
integrityOf_depositIntoYieldVault	Verified	`nonDeposited = depositIntoYieldVault(assets)` *then* `nonDeposited <= assets.`	Report
integrityOf_simulateWithdraw_force	Verified	`_simulateWithdraw(x, true) == x.`	Report
previewDepositCorrectness_strict **previewMintCorrectness_strict** **previewRedeemCorrectness_strict** **previewWithdrawCorrectness_strict**	Verified	*Preview methods provide exact information.*	Report
conversionOfZero	Verified	*convertTo(0) == 0. (Both* `convertToAssets` *and* `converToShares`*.)*	Report
convertToAssetsWeakAdditivity **convertToSharesWeakAdditivity**	Verified	*convertTo(A) + convertTo(B) <= convertTo(A+B) (Both* `convertToAssets` *and* `converToShares`*.)*	Report

conversionWeakMonotonicity	Verified	*A < B then convertTo(A) <= convertTo(B) (Both* `convertToAssets` *and* `convertToShares`*.)*	Report
conversionWeakIntegrity	Verified	`convertToShares(convertToAssets(X)) <= X`	Report
depositMonotonicity	Verified	*A < B, then* `deposit(A)` *gives less or equal shares than* `deposit(B)`	Report
zeroDepositZeroShares	Verified	`deposit(x) == 0` *if and only if* `x == 0.`	Report

P-04. Risk assessment

Status: Verified	

Rule Name	Status	Description	Link to rule report
onlyAllowedMethodsMayChangeMigrationFeesFund	Verified	*Only specified methods may change* `migrationFeesFund`*.*	[Report](#)
onlyAllowedMethodsMayChangeTotalAssets	Verified	*Only specified methods may change* `totalAssets`*.*	[Report](#)
onlyAllowedMethodsMayChangeTotalSupply	Verified	*Only specified methods may change* `totalSupply`*.*	[Report](#)
onlyAllowedMethodsMayChangeStakedAssets	Verified	*Only specified methods may change* `stakedAssets`*.*	[Report](#)
noDynamicCalls	Verified	*There are no dynamic calls to untrusted contracts.*	[Report](#)
underlyingCannotChange	Verified	*address of* `asset()` *never changes.*	[Report](#)
dustFavorsTheHouse	Verified	`redeem(deposit(x))` *doesn't decrease the balance of the contract.*	[Report](#)
redeemingAllValidity	Verified	*After redeeming the entire balance, the user's balance is zero.*	[Report](#)

onlyContributionMethodsReduceAssets **contributingProducesShares**	Verified	*Only specified methods may decrease the user's balance.* *When a user contributes assets, they are given shares.*	[Report]()
reclaimingProducesAssets	Verified	*When calling* `withdraw` *or* `redeem` *with* `receiver` *and* `owner`*,* `owner’s` *shares go down if and only if* `receiver’s` *assets go up.*	[Report]()

GenericNativeConverter and MigrationManager

Module Properties

We verified that the GenericNativeConverter is always solvent and two more important properties about *backingOnLayerY* and *nonMigratableBacking.*

P-05. Risk assessment	
Status: Verified after fix	

Rule Name	Status	Description	Link to rule report
converterSolvency	Verified	*The balance of the converter is at least* `backingOnLayerY`*.*	Report
backingMoreThanSupply	Verified	`backingOnLayerY` *is at least* `customToken.TotalSupply` *(minus bridged assets).*	Report
nonMigratableBackingPercentageLT_E18	Verified	`nonMigratableBackingPercentage` *cannot exceed* `10^18`*.*	Report
nonMigratableBackingAlwaysPresent	Verified after fix Reported issue I-02	`backingOnLayerY` *can't go below* `nonMigratableBacking`*.*	Original report Report after fix
nonMigratableBackingAlwaysPresent_margin1	Verified	`backingOnLayerY` *+ 1 can't go below* `nonMigratableBacking`*.*	Report
onMsgReceived_doesntAlwaysRevert	Verified	`MigrationManager.onMsgReceived doesn't always revert`	Report

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.