SOLIDIFIED

Audit Report for Aztec - November 2022

Summary

Audit Report prepared by Solidified covering the Aztec protocol Ethereum Bridge contract for Liquity Trove Bridge.

The following report covers the **Liquity Trove Bridge**.

This audit is a re-audit of the first Liquity bridge audit. The Aztec team changed the behavior of the bridge in the repayment case.

Process and Delivery

Independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 21 November 2022.

Audited Files

The source code has been supplied in the form of one public Github repository.

https://github.com/aztecProtocol/aztec-connect-bridges/

Commit Hash: `46af4186f21cd1925d41fd0275883be18755d212`

```
src
|-- bridges
|   -- liquity
      |-- TroveBridge.sol
```

Intended Behavior

Aztec Connect Bridge for interacting with Liquity's troves.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than in a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Issue #	Description	Severity	Status
1.	Deposit ETH Attack can revert every repayment bridge call with an underflow	Critical	Resolved
2.	Information Notes	Notes	Resolved

Critical Issues

1. Deposit ETH Attack can revert every repayment bridge call with an underflow

In case a Trove `Redistribution` happened an attacker could send ETH to the bridge contract to perform an attack which would cause the bridge transaction to revert with an math-underflow.

This would result in a revert of all repay interactions.
(As a result locked ETH in the Trove would be lost)

Each Trove has a `debt` and a `coll` value.

In a Trove redistribution event `debt` and `coll` are both increased. Let say the collateral `coll` is increased by an amount `c'`.

The actual value of `c'` depends on the liquidated Trove size. However the bridge Trove could just be partially affected and the actual value of `c'` could be a tiny ETH amount worth a few dollars.

An attacker could just send the following ETH amount to the `TroveBridge` contract after the `redistribution`:

```
attackAmount = c' + 1
```

All repay transactions would revert. For a smaller attackAmount it would depend on the bridge `_totalInputValue` value if it is enough for an underflow.

The problem lies in the usage of ETH balance in `_repayWithCollateral` method:

```
collateralReturned = address(this).balance;
```

This happens after the Uniswap repayment so the actual ETH balance would be:

```
// correct collateralReturned + attack Amount
address(this).balance = collateralReturned + attackAmount
```

Even if we assume the entire trove collateral has been withdrawn from the Trove, the actual collateral sold to Uniswap will be just `c'` at maximum. (Actually a bit less because of the 110% over-collateralization)

`collToWithdraw = coll + c'`

Therefore, we can define the actual collateral returned as

`collateralReturned=collToWithdraw-c'`

```
//                actual ETH balance of the contract
//                = collateralReturned + attackAmount
//                = collToWithdraw-c' + attackAmount
//                = collToWithdraw-c' + c' + 1
//                = collToWithdraw + 1
collateralReturned = address(this).balance;
...
```

Subtraction Underflow Revert

```
//                      = subtraction underflow
//                      = collToWithdraw - (collToWithdraw + 1);
uint256 collateralSold  = collToWithdraw - collateralReturned;
```

If the entire collateral is not removed from the Trove the underflow would be higher than 1 Wei.

Recommendation:
Store the `address(this).balance` in a local variable before the Uniswap interaction and calculate the delta between balance afterwards. This delta amount should be used for `collateralReturned` calculation.

Resolved:
In the following commit `89129e99529b0095310d52396a68edf251043a9b`

Major Issues

No issues found

Minor Issues

No issues found

2. Informational Notes

Variable Naming Improvement suggestions

In order to increase the code clarity consider renaming `collateralSold` to `collateralSoldToUniswap`, and `maxCost` to `maxCostInETH`.

Check for inputAssetB == None

In the `convert` function a check for `inputAssetB == None` could be added to ensure correct usage.

```
else if (
            _inputAssetA.erc20Address == address(this) &&
_outputAssetA.assetType == AztecTypes.AztecAssetType.ETH
        )
```

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Aztec Protocol or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors from legal and financial liability.

Oak Security GmbH