certora

Security Assessment Report

Relend Network

February 2025

Prepared for Relend Network

Table of content

Project Summary **3**
Project Scope 3
Project Overview 3
Protocol Overview 3
Findings Summary 5
Severity Matrix 5
Detailed Findings **6**
Low Severity Issues 7
L-01 MorphoBank cannot rescue excess collateral from Morpho 7
L-02 Refunds from bridges can be stolen when delivered to Dummy 8
Informational Severity Issues **10**
I-01. Floating pragma 10
I-02. RelendWTokenL1 code is unnecessarily imported by MorphoBank 10
I-03. Initializing variables to zero is redundant 11
I-04. Remove magic numbers 11
Disclaimer **12**
About Certora **12**

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Relend Network	https://github.com/backstop-protocol/ERC-7770/tree/certoraAudit/src/L1	884fb467f149b4f21ca89a2aedcad44c7331ae15	EVM

Project Overview

This document describes the verification of **Relend Network** using manual code review findings. The work was undertaken from **3 Feb** to **7 Feb 2025**

The following contract list is included in our scope:

backstop-protocol/ERC-7770/tree/certoraAudit/src/L1/*

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

Relend network deploys a new stablecoin for every partner L2. The stablecoin is deployed on Ethereum mainnet, and it is nothing more than a USDC wrapper, augmented with additional permissioned mint (fractionalReserveMint) and burn (fractionalReserveBurn) functions.

Users who wrapped their USDC to xxxUSD (and typically bridged it to XXX L2) provided USDC liquidity for other xxxUSD users (who bridged their xxxUSD back to the ethereum mainnet) to redeem their xxxUSD back to USDC. When the USDC liquidity in the xxxUSD contract is low (because xxxUSD borrowers decided to redeem), a second price stability module is kicking in.

The second price stability module is called the Morpho Bank (MorphoBank contract). Relend network is managing a normal morpho vault (https://docs.morpho.org/morpho-vaults/contracts/overview/) and supplying over $20m of USDC to Morpho's market. The relend network vault will create and provide liquidity upon demand to a premissioned-wrapped-xxxUSD/USDC market on morpho (first asset is collateral, second is debt asset), where it could borrow USDC vs PWxxxUSD in a 1:1 ratio and with zero interest rate. Upon demand (low USDC liquidity for xxxUSD on mainnet), the bank will take a USDC flashloan from morpho, will wrap it to PWxxxUSD, and will borrow USDC from morpho to repay the flashloan.

certora

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	2	2	0
Informational	4	4	1
Total	6	6	1

Severity Matrix

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

certora

Detailed Findings

ID	Title	Severity	Status
L-01	MorphoBank cannot rescue excess collateral from Morpho	Low	{Not yet fixed}
L-02	Refunds from bridges can be stolen when delivered to Dummy	Low	{Not yet fixed}

Low Severity Issues

L-01 MorphoBank cannot rescue excess collateral from Morho

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: MorphoBank.sol	Status: Acknowledged	Violated Property: -

Description: `MorphoBank` is designed to wrap/unwrap stablecoin tokens locked in the Morpho lending protocol via a top-up/top-down mechanism. This mechanism is a 1:1 exchange and works under the assumption that collateral (wrapped) tokens locked in Morpho always match the amount of loan (unwrapped/underlying) tokens borrowed.

This assumption however does not always holds true. For example, after a liquidation in Morpho (which can be admitted if the team changes the price oracle setting), some collateral may be left over without corresponding debt. The same could happen in the case of an external actor donating a `repay` for the `MorphoBank`'s debt to Morpho - an operation that on Morpho can be done by anyone, not only the debtor address.

Exploit Scenario: When, in the cases described above, `MorphoBank` has an excess collateral, it won't be able to make use of it.

Recommendations: We recommend adding a permissioned function to `MorphoBank` that allows admins to rescue collateral that `MorphoBank` has in excess of its debt.

Customer's response: Acknowledged and won't fix. This is a nice optimization but it does not affect the solvency of the stable coin or the morpho market. And it is not expected to strictly produce a better result in a normal running flow.

Fix Review: Issue Acknowledged.

certora

L-02 Refunds from bridges can be stolen when delivered to Dummy

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: RelendWTokenL1.sol	Status: Acknowledged	Violated Property: Funds returned by bridges should be safely available to the sender

Description: The `RelendWTokenL1` contract allows bridging its tokens to L2s via the function `depositForAndCall`. This function is designed to be as generic as possible to allow integration with a variety of bridges without requiring any change in on-chain code, in fact allowing callers to provide their logic via the `msg.value`, `_callTarget`, `_callData` they provide:

```
JavaScript
File: RelendWTokenL1.sol
12: contract Dummy {
13:     using Address for address;
14:
15:     function doArbitraryCall(address target, bytes calldata data) payable external {
16:         target.functionCallWithValue(data, msg.value);
17:     }
18: }
19:
---
63:     function depositForAndCall(address _account, uint256 _value, address _callTarget,
bytes calldata _callData) external payable returns (bool) {
64:         require(ERC20Wrapper.depositFor(_account, _value), "depositForAndCall: depositFor
failed");
65:         dummy.doArbitraryCall{value: msg.value}(_callTarget, _callData);
66:
67:         return true;
68:     }
```

The problem with this approach is that some bridges, like the Starknet or the Taiko bridge, fund the address that originally called them in case the message bridging failed or was cancelled.

When this address is an open contract, controllable by anyone at any time, like `Dummy` or a `Multicall` that `Dummy` can use to multiplex calls, returned funds can be stolen by anyone.

Exploit Scenario: Alice bridges some tokens to chain X via `RelendWTokenL1.depositForAndCall`; the bridge fails to deliver these tokens, and sends them back to the address that interacted with the bridge, that is the `Dummy` contract. Immediately afterwards, anyone can call `Dummy.doArbitraryCall(...)` to steal these tokens from Alice.

Recommendations: We recommend using coded logic for bridging, which can be whitelisted and pluggable for extra flexibility. Alternatively, calls can be routed through a contract created ad-hoc that can be controlled only by `RelendWTokenL1` during the deposit and the only the depositor afterwards. This second approach could however expensive in gas.

We also recommend conducting a thorough security review for the bridging logic, regardless of whether it's generated off-chain as it is now, or it's coded on-chain.

Customer's response: This is useful information and we will communicate it to the bridge teams that integrate us. The initial feedback from starknet and takio is that cancellation is only possible upon bridge failure, which is not very likely to happen, but nevertheless, we will start by asking the user to do two separate transactions. One to wrap the USDC, and another to bridge it.

Fix Review: Issue Acknowledged.

Informational Severity Issues

I-01. Floating pragma

Description: All of the Solidity files in scope are headed with the following pragma statement

```
JavaScript
pragma solidity ^0.8.13;
```

This notation is generally discouraged practice and is standardized as a weakness (SWC-103) because the compilation outcome is not deterministic and cannot be guaranteed unaffected by Solidity compiler and/or optimizer bugs.

Recommendations: We recommend pointing to a fixed Solidity version known to be stable.

Customer's response: Fixed with this commit https://github.com/backstop-protocol/ERC-7770/commit/6b0af1fd6a7bd5822a0a7bebb0dcbdbe36d2c92d

Fix Review: Fixed Appropriately.

I-02. RelendWTokenL1 code is unnecessarily imported by MorphoBank

Description: The `MorphoBank` contract interacts with the `RelendWTokenL1` contract via external calls, to read its underlying ERC20 and wrap and unwrap amounts of this token.

It however does so by unnecessarily importing the whole `RelendWTokenL1` contract, not only its interface, and this is inefficient because more code ends up being compiled than necessary.

Recommendation: Consider adding a `IERC20wrapper` interface to be imported and used by `MorphoBank` in place of `RelendWTokenL1`.

Customer's response: Acknowledged but won't fix.

Fix Review: Acknowledged.

I-03. Initializing variables to zero is redundant

Description: Solidity automatically assigns new variables to zero. Therefore, assigning them a zero value is redundant.

Recommendation: We recommend changing RelendWTokenL1.sol line 21 to `uint256 private _totalBorrowedSupply;` and removing MorphoBank.sol line 48 where `marketParams.irm` is unnecessarily set.

Customer's response: We init total borrowed supply to 0 for good order and believe it is more readable. In line 48 we believe it is imperative to explicitly show the irm is set to 0, as otherwise the reader won't notice it.

Fix Review: Issue acknowledged, won't fix.

I-04. Remove magic numbers

Description: Magic numbers decrease readability and overall code quality.

Recommendation: We recommend making such numbers constants and documenting the reasoning behind their existence.

Replace `marketParams.oracle` calculation with a constant (*`1021408163265306122448979591836734693`*) and document how such value was arrived at.

```
marketParams.oracle = address(new FixedPriceOracle(uint256(1e36 * 100) /
uint256(98) + uint256(1e33), _oracleOwner));
```

Replace `0.98e18` with a constant and document the reasoning behind it.
`marketParams.lltv = 0.98e18;`

Customer's response: Acknowledged but won't fix. The reasoning behind the oracle price is documented in the code. We prefer not to define constants for numbers that are only used once in the code.
Fix Review: Issue acknowledged, won't fix.

certora

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.