`

Immutable zkEVM Bridge

Defensive Fuzzing Report

09/10/2024

Prepared By:
0xScourgedev | Lead Fuzzing Specialist
0xscourgedev@perimetersec.io

Table of Contents

About Perimeter

Perimeter's mission is to deliver the highest quality fuzzing services to protocols by uniting the world's foremost fuzzing specialists. We possess extensive expertise in fuzzing a diverse range of protocols, from smaller, niche protocols to some of the largest and most complex in DeFi.

In order to deliver on our mission, we have developed the most advanced scaffolding and libraries, enabling us to create highly sophisticated fuzzing suites tailored to meet the unique challenges of each protocol.

Learn more about us at perimetersec.io

Services Provided

Perimeter has successfully delivered a comprehensive suite of services that include:

- **Fuzzing Suite Development:** Design and implement a stateful fuzzing suite using Echidna and Medusa. This suite will be tailor-made for the protocol and contracts in scope. The completed fuzzing suite can later be integrated into the testing suite to serve long-term security needs.
- **Invariant Replication:** Replicate some of the foundry invariants to be included in the fuzzing suite.
- **Invariant Testing Assurance:** Guarantee that each invariant implemented will be tested no fewer than 50,000,000 instances, ensuring thorough validation and reliability.
- **Proof-of-Concept Development:** Develop a corresponding Proof-of-Concept (PoC) for each finding and assertion/property counterexample identified, to demonstrate potential vulnerabilities and their implications.
- **Comprehensive Final Report:** Create a detailed final report that will include all findings, along with their corresponding PoCs. This report will also detail the invariants tested, their run status, and the number of runs, providing a comprehensive overview of the engagement's outcomes.

Files in Scope

The engagement will be focused on the files listed below, acquired from commit [0a56c7cf3675ff9c71010d2d768da4bf1dd7f204](0a56c7cf3675ff9c71010d2d768da4bf1dd7f204).

```
src
├── child
│   ├── ChildERC20Bridge.sol
│   ├── ChildERC20.sol
│   └── WIMX.sol
└── root
    ├── flowrate
    │   ├── FlowRateDetection.sol
    │   ├── FlowRateWithdrawalQueue.sol
    │   └── RootERC20BridgeFlowRate.sol
    └── RootERC20Bridge.sol
```

Files Out of Scope

Files outside the scope were not directly considered in achieving the target. However, since many of these files are utilized by those within the scope, a significant portion was indirectly covered.

Methodology

The primary objective of this engagement was to replicate the invariants from the existing Foundry suite into an Echidna/Medusa invariant suite for long-term use.

The Echidna and Medusa fuzzers currently have much stronger capabilities of detecting broken invariants compared to the Foundry fuzzer. As a result, developing the fuzzing suite for Echidna and Medusa is extremely beneficial for the long-term security of the protocol.

Additionally, the engagement aimed to incorporate numerous new invariants, many related to the root bridge's flow rate, to conduct thorough stress testing of the contracts and ensure high coverage rates for the invariant suite.

The invariant suite developed for this engagement tests each side of the bridge individually with a superset of expected values to thoroughly test for edge cases and to reach additional coverage. Fuzzing across the root bridge and the child bridge is out of scope of this engagement and can be addressed in a follow-up engagement.

Invariants

We created many tests to verify the correctness of **66** invariants described in the table below. During the execution phase, these invariants were assessed for 610M+ runs on Echidna and 800M+ calls on Medusa for a total of **1.41B+** runs.

The table below lists all invariants that are part of this engagement.

Invariant	Description	Tested	Passed	# Runs
FLRT-01	If imxCumulativeDepositLimit is not 0, the rootIMX token balance of the root bridge should always be less than or equal to imxCumulativeDepositLimit	✅	✅	1.41B+
FLRT-02	If the withdrawal amount is greater than the largeTransferThreshold, the withdrawal must be added to the user's withdrawal queue	✅	✅	1.41B+
FLRT-03	If the bucket capacity for a token is 0, then any withdrawal of that token must be added to the user's withdrawal queue	✅	✅	1.41B+
FLRT-04	If withdrawalQueueActivated is true, then any withdrawal of any token must be added to the user's withdrawal queue	✅	✅	1.41B+
FLRT-05	If finaliseQueuedWithdrawal is successfully called, then the queued withdrawal initiated timestamp plus the withdrawal delay must be less than or equal to the current block timestamp	✅	✅	1.41B+
FLRT-06	After a successful withdrawal of a non-zero amount on the root bridge, if the bucket capacity is not zero, then the token bucket depth is never equal to the bucket capacity	✅	✅	1.41B+
FLRT-07	After a successful withdraw on the root bridge, if the withdrawn amount is less than the minimum of the refill rate multiplied by the time elapsed since last withdrawal and the bucket capacity minus the prior real bucket depth, then the bucket depth strictly increases	✅	✅	1.41B+
FLRT-08	After a successful withdraw on the root bridge, the bucket depth is always less than or equal to the bucket capacity	✅	✅	1.41B+

Invariant	Description	Tested	Passed	# Runs
FLRT-09	When the withdrawalQueue is not activated, after a successful withdrawal of an amount less than the largeTransferThreshold on the root bridge and a withdrawal was added to the user's withdrawal queue, the bucket depth is always 0	✅	✅	1.41B+
FLRT-10	When the withdrawalQueue is not activated, after a successful withdrawal of an amount less than the largeTransferThreshold on the root bridge, if the withdrawn amount is greater or equal to minimum of the refill rate multiplied by the time elapsed since last withdrawal plus the prior real bucket depth and the bucket capacity, then the withdrawal must be added to the user's withdrawal queue	✅	✅	1.41B+
FLRT-11	If the depth of the token bucket is non-zero before a successful withdrawal on the root bridge and the depth of the token bucket is zero after the withdrawal, then the withdrawal queue must be activated	✅	✅	1.41B+
FLRT-12	After each successful withdrawal on the root bridge for a token with a non-zero capacity, the bucket refillTime is always updated to the current block timestamp	✅	✅	1.41B+
SPLY-01	The sum of the token balances of each user and the bridges should be exactly equal to the token total supply	✅	✅	1.41B+
SPLY-02	The total supply of root WETH must be decreased by the amount of WETH deposited by the user	✅	✅	1.41B+
SPLY-03	The total supply of child ERC20s must be increased by exactly the amount of tokens deposited by the user	✅	✅	1.41B+
SPLY-04	The total supply of child ERC20s must be decreased by exactly the amount of tokens withdrawn by the user	✅	✅	1.41B+

Invariant	Description	Tested	Passed	# Runs
BAL-01	The WETH balance of the root bridge should always be 0	✅	✅	1.41B+
BAL-02	The WIMX balance of the child bridge should always be 0	✅	✅	1.41B+
BAL-03	The native balance of the root adaptor increases by exactly the gas fees paid by the users	✅	✅	1.41B+
BAL-04	The native balance of the child adaptor increases by exactly the gas fees paid by the users	✅	✅	1.41B+
BAL-05	The token balance, excluding WETH, of the root bridge increases by exactly the amount of tokens deposited by the user	✅	✅	1.41B+
BAL-06	The native balance of the root bridge increases by exactly the amount of WETH deposited by the user	✅	✅	1.41B+
BAL-07	When depositing ETH, the native balance of the root bridge increases by exactly the amount deposited by the user, minus the gas fees	✅	✅	1.41B+
BAL-08	When withdrawing IMX, the native balance of the child bridge increases by exactly the amount withdrawn by the user, minus the gas fees	✅	✅	1.41B+
BAL-09	The native balance of the child bridge decreases by exactly the amount of root ERC20 IMX deposited by the user	✅	✅	1.41B+
RTREV-01	activateWithdrawalQueue never reverts	✅	✅	1.41B+
RTREV-02	deactivateWithdrawalQueue never reverts	✅	✅	1.41B+
RTREV-03	deposit never reverts unexpectedly	✅	✅	1.41B+
RTREV-04	depositETH never reverts unexpectedly	✅	✅	1.41B+
RTREV-05	depositTo never reverts unexpectedly	✅	✅	1.41B+
RTREV-06	depositToETH never reverts unexpectedly	✅	✅	1.41B+

Invariant	Description	Tested	Passed	# Runs
RTREV-07	finaliseQueuedWithdrawal never reverts unexpectedly	✅	✅	1.41B+
RTREV-08	finaliseQueuedWithdrawalAggregated never reverts unexpectedly	✅	✅	1.41B+
RTREV-09	onMessageReceive for the root bridge never reverts unexpectedly	✅	✅	1.41B+
RTREV-10	pause for the root bridge never reverts	✅	✅	1.41B+
RTREV-11	setRateControlThreshold never reverts	✅	✅	1.41B+
RTREV-12	setWithdrawalDelay never reverts	✅	✅	1.41B+
RTREV-13	unpause for the root bridge never reverts	✅	✅	1.41B+
RTREV-14	updateImxCumulativeDepositLimit never reverts unexpectedly	✅	✅	1.41B+
CLDREV-01	onMessageReceive for the child bridge never reverts unexpectedly	✅	✅	1.41B+
CLDREV-02	pause for the child bridge never reverts	✅	✅	1.41B+
CLDREV-03	unpause for the child bridge never reverts	✅	✅	1.41B+
CLDREV-04	withdraw never reverts unexpectedly	✅	✅	1.41B+
CLDREV-05	withdrawETH never reverts unexpectedly	✅	✅	1.41B+
CLDREV-06	withdrawETHTo never reverts unexpectedly	✅	✅	1.41B+
CLDREV-07	withdrawIMX never reverts unexpectedly	✅	✅	1.41B+
CLDREV-08	withdrawIMXTo never reverts unexpectedly	✅	✅	1.41B+
CLDREV-09	withdrawTo never reverts unexpectedly	✅	✅	1.41B+
CLDREV-10	withdrawWIMX never reverts unexpectedly	✅	✅	1.41B+
CLDREV-11	withdrawWIMXTo never reverts unexpectedly	✅	✅	1.41B+

Invariant	Description	Tested	Passed	# Runs
PAUSE-01	deposit always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-02	depositETH always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-03	depositTo always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-04	depositToETH always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-05	finaliseQueuedWithdrawal always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-06	finaliseQueuedWithdrawalAggregated always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-07	onMessageReceive for the root bridge always reverts when the root bridge is paused	✅	✅	1.41B+
PAUSE-08	withdraw always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-09	withdrawETH always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-10	withdrawETHTo always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-11	withdrawIMX always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-12	withdrawIMXTo always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-13	withdrawTo always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-14	withdrawWIMX always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-15	withdrawWIMXTo always reverts when the child bridge is paused	✅	✅	1.41B+
PAUSE-16	onMessageReceive for the child bridge always reverts when the child bridge is paused	✅	✅	1.41B+

LOW-01: Missing root IMX token mapping is misleading

Severity

Low

Description

`ChildERC20Bridge::rootTokenToChildToken` does not map the root ERC20 IMX token to `NATIVE_IMX`, which can cause confusion, as it can lead users to believe the root ERC20 IMX token does not have a child chain equivalent as it returns `address(0)`.

Additionally, it's inconsistent behavior with `RootERC20Bridge.sol`, as it correctly maps `rootTokenToChildToken` with `NATIVE_ETH`.

Recommendation

Add the mappings for `NATIVE_IMX` to the `ChildERC20Bridge::initialize` function.

Response

The team acknowledges this finding. While it has no impact on the bridge's functionality, as IMX deposits and withdrawals do not rely on the `rootTokenToChildToken` mapping, it is true this is inconsistent with how we treat `NATIVE_ETH` and can thus cause confusion. We plan to address this in an upcoming upgrade.

Disclaimer

All activities conducted by Perimeter in connection with this project were carried out in accordance with the terms outlined in a Statement of Work and an agreed-upon project plan, as set forth in a proposal document delivered prior to the commencement of the project.

Security assessment projects are subject to time limitations, and as such, the findings presented in this report should not be interpreted as an exhaustive or comprehensive identification of all security issues, vulnerabilities, or defects within the target codebase. Perimeter makes no representations or warranties that the target codebase is free from defects.

Furthermore, this report is not intended to be, and should not be construed as, investment advice or a recommendation to participate in any financial transactions. The content herein does not constitute endorsements or recommendations for any financial decisions, securities, or investment strategies.