Security Audit Report

Axelar evm-native-wrapper
Ethereum smart contract

by ARDA

on July 5, 2026

Table of Contents

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's design, which are a subset of all the possible risks. **No inherent risk doesn't mean no risk.** It only means that no risk inherent to the code's design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the upgradability of the code might also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Summary

Blockchain: Ethereum

Initial scope

- **Repository:** https://github.com/axelarnetwork/axelar-app
- **Commit:** `fce510ca2d02752be45449397091c4855874b4bb`
- **Path(s):** `./apps/contracts/contracts/NativeWrapper.sol`

Final scope

- **Repository:** https://github.com/axelarnetwork/axelar-app
- **Commit:** `62b87d4cfbe976b0e04ec4fdb43d249e42f94aae`
- **Path(s):** `./apps/contracts/contracts/NativeWrapper.sol`

0 inherent risk in the final scope

0 issue in the final scope

1 issue reported in the initial scope and 0 remaining in the final scope.

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	1	0	0	0	0	0
Minor	0	0	0	0	0	0

Code Issues & Recommendations

C1: wrapAndBridge accepts any tokenId but should only accept canonical WETH tokenID

Severity: Medium **Status:** Fixed

Location

```
apps/contracts/contracts/NativeWrapper.sol
   wrapAndBridge
```

Description

Current behavior: the constructor stores `weth` and `its` as immutables (NativeWrapper.sol:30-33), but `wrapAndBridge` (NativeWrapper.sol:47-75) accepts an arbitrary `tokenId` from the caller and forwards it to `its.interchainTransfer` without verifying that `tokenId` is the canonical ITS `tokenId` associated with WETH.

Consequently, if the caller passes a `tokenId` whose ITS Token Manager **is not the canonical WETH** `tokenId`, then this Token Manager might not behave as expected on the destination chain, e.g. the user could receive a different unwanted token, or not tokens at all.

Expected behavior: the contract should enforce that the `tokenId` used in ITS is the canonical WETH `tokenId`.

Worst consequence: a user calls `wrapAndBridge` with any `tokenId` registered in ITS against the same `weth` address but not equal to the intended WETH `tokenId` and loses the entire bridged amount: the WETH is bridged under the wrong `tokenId` to a different destination token contract, with no path to recover the funds from this contract.

Recommendation

We suggest removing the `tokenId` argument from `wrapAndBridge`, and instead making `tokenId` into an immutable variable, set at deployment by fetching the WETH canonical token from ITS.

We can further add two unit tests covering (1) the success path where `tokenId` corresponds to WETH canonical token, and (2) the failure case for any other `tokenId` value.