Security Audit Report

Axelar mvx-interchain-token-service-proxy

MultiversX smart contract

by ARDA

on April 21, 2025

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.

Audit Summary

Initial scope

- **Repository:** https://github.com/multiversx/sc-axelar-cgp-rs
- **Commit:** `ac33c60b737ef0c212adf6e14646815b777e35d4`
- **MultiversX smart contract path:** ./interchain-token-service-proxy/

Final scope

- **Repository:** https://github.com/multiversx/sc-axelar-cgp-rs
- **Commit:** `b863a1ba7fe8180e63961f721a63c6d53d818137`
- **MultiversX smart contract path:** ./interchain-token-service-proxy/

1 inherent risk in the final scope

0 issue in the final scope

3 issues reported in the initial scope and 0 remaining in the final scope:

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

Inherent Risks

R1: Users can’t get back the tokens from smart contract calls which are not successfully executed.

This is because there is no refunding mechanism implemented in the Proxy.

In particular, if a user’s incoming interchain transfer going through the Proxy is never successfully executed, then the user won’t ever get back his tokens.

Code Issues & Recommendations

C1: Failed smart contract calls can’t be retried

Severity: Critical **Status:** Fixed

Location

`interchain-token-service-proxy/src/lib.rs`

Description

Current behavior: When the Proxy processes an incoming interchain transfer with smart contract call, if the call fails, then it can’t be retried.

More precisely, when the ITS forwards an incoming interchain token transfer to the Proxy, in order to call the endpoint `executeWithInterchainToken` of the destination smart contract, the call is performed asynchronously. Then, if the call fails, in the callback the user’s tokens are simply kept in the Proxy, and there is no endpoint to retry the call.

In particular, the call to the destination smart contract might fail for any reason, e.g. due to a temporary logical error in the destination smart contract or due to insufficient gas. However, after one failure, the Proxy does not allow to perform the call again, even under conditions where it could in principle be executed successfully.

Expected behavior: Users should be allowed to retry executing incoming interchain transfers with smart contract calls, as it would significantly increase the chances that eventually the call will be successfully executed, and that the users’ tokens will be spent as intended. Indeed, one failure of a smart contract call at a specific time and under specific conditions does not mean that the call can’t be successful at a later time and under different conditions.

Worst consequence: A user loses the tokens of his interchain transfer because the smart contract call failed only once.

Moreover, when combined with another issue of this report, C2: The owner can't protect its users against too small gas for smart contract calls, **this issue allows an attacker to stuck the tokens of users' incoming interchain transfers** going through the Proxy. For this, the malicious actor would execute users' interchain transfers with insufficient gas for the smart contract call.

Example: An interchain transfer of 10000 USDC is made and forwarded to the Proxy, in order to perform a swap on xExchange for EGLD, with a slippage protection. At the time this call is executed, it fails as it would lead to a too small amount of EGLD, thanks to the slippage protection. A few hours later, the user observes that a swap would now pass the slippage protection, however he can't attempt to re-execute the swap, and his 10000 USDC are stuck in the smart contract.

Recommendation

We recommend allowing users retrying the execution of failed incoming interchain transfers with smart contract calls.

For this:

- We modify the storage `failed_calls` such that it records all the necessary information of a call:

```
fn failed_calls(
  source_chain: ManagedBuffer,
  message_id: ManagedBuffer,
  source_address: ManagedBuffer,
  data: ManagedBuffer,
  token_id: TokenId
) -> SingleValueMapper<(EgldOrEsdtTokenIdentifier, BigUint)>;
```

- In `execute_with_interchain_token`, if the caller is not ITS, we require that the call is recorded in `failed_calls`, and that no tokens are received. Then, we clear `failed_calls` and execute the call.

C2: The owner can't protect its users against too small gas for smart contract calls

Severity: Medium **Status:** Fixed

Location

`interchain-token-service-proxy/src/lib.rs`

Description

Current behavior: The owner of the Proxy can't protect its users against too small gas for calls to the endpoint `executeWithInterchainToken` of the destination smart contract.

Indeed, the gas for this call can be arbitrary, as it is deduced from the gas provided by the caller, which can be arbitrary. If the gas is too low, it would make the call fail.

Expected behavior: The owner should be able to guarantee that enough gas is provided to execute users' smart contract calls.

Worst consequence: Let's assume that failed executions can be retried, i.e. that the issue C1: Failed smart contract calls can't be retried has been resolved. Even in this case, an attacker might prevent an incoming interchain transfer with a call to a smart contract on a different shard from being executed over an arbitrarily long period of time. For this, he would execute the call with insufficient gas, and repeat this each time the callback is reached.

Note however that, in order to perform such an attack, the attacker would need to be the first to re-execute the interchain transfer each time the callback is reached.

Recommendation

We recommend introducing a new storage `min_gas_for_execution` that the owner would set in `init` and through a dedicated owner endpoint.

Then, in the endpoint `execute_with_interchain_token`, we would start by verifying that the gas left is at least `min_gas_for_execution`.

C3: Typo in comment

Severity: Minor **Status:** Fixed

Location

`interchain-token-service-proxy/src/lib.rs`
`execute_callback`

Description

In the method `execute_callback`, there is a typo “your a” in the following comment:

```
// that would be relevant for a your dApp
```

Recommendation

We recommend correcting the comment.