Security Audit Report

Axelar solana-interchain-token-service (3)

Solana program

by ARDA

on April 15, 2026

Table of Contents

Disclaimer **3**

Terminology **4**

Objective **4**

Summary **5**

Inherent Risks **6**

Code Issues & Recommendations **10**

C1: Tokens of inbound transfers with CPI are stuck and destination program can’t spend them 10

C2: Misleading parameter names 11

C3: Unused error variants in RolesError 12

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: Solana

Initial scope

- **Repository:** https://github.com/axelarnetwork/axelar-amplifier-solana
- **Commit:** `30e4195a839ded2d7a695b2e9e71e06242a43b58`
- **Path(s):**
  `./programs/solana-axelar-its`

Final scope

- **Repository:** https://github.com/axelarnetwork/axelar-amplifier-solana
- **Commit:** `3e7a6f81ea41bc1eafb78dd10acbdbb589bd1a00`
- **Path(s):**
  `./programs/solana-axelar-its`

4 inherent risks 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	0	0	0	0	0	0
Major	1	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	2	0	0	0	0	0

Inherent Risks

R1: Users’ outgoing interchain transfers might never be successfully executed on the destination blockchain and they will never be refunded on Solana.

This is because, for reasons explained below, outgoing interchain transfers might either (1) never be executed, or (2) be incorrectly executed, or (3) systematically fail to be executed. Moreover, in such cases the tokens would be lost as there is no refunding mechanism implemented on Solana.

1) An outgoing interchain transfer might never be executed if for any reason the Axelar network does not forward it to the destination blockchain.

2) An outgoing interchain transfer might be incorrectly executed if the transfer's information is altered by the Axelar network while being forwarded to the destination blockchain.

Example: The ITS Hub on the Axelar blockchain modifies the amount of the transfer based on scaling rules which depend on the token decimals (both on the source and destination blockchains) and on the blockchains’ requirements (e.g. the maximal allowed integers). In particular, truncations might lead to smaller transferred amounts than expected.

Moreover, the decimals themselves might have changed since they were registered (once and for all) the first time in ITS. Namely, this is possible when a mint from the Extension Token Program is closed on Solana (which is only allowed if the mint supply has reached 0), and is re-created afterwards with different decimals.

3) An outgoing interchain transfer might systematically fail to be executed on the destination blockchain, i.e. even if it is possible to re-try the execution, it would never succeed. There are two main types of failures:

3-a) Failures due to user errors in the interchain transfer's information.

Examples:

- The transfer's data (e.g. the destination address or smart contract call) is in a format incompatible with the destination blockchain.

- There is no Token Manager deployed on the destination blockchain for the token being transferred.

3-b) Failures that the user did not expect or could not anticipate.

Examples:

- The Token Manager of a custom token on the destination blockchain is of type *"Lock/Unlock"* and has an insufficient reserve of tokens. This is impossible to anticipate: the reserve of the Token Manager on the destination blockchain might have been sufficient at the time the interchain transfer was registered on Solana, but might have become too small by the time the interchain transfer arrives at destination.
- The Token Manager is of type *"Mint/Burn"* and has lost its minting role by the time the interchain transfer arrives at destination.
- The admins of the Token Manager on the destination blockchain have the ability to limit the amounts of tokens that can be transferred to that blockchain. Therefore, they could completely block interchain transfers.
- The destination address is a smart contract that is upgraded while the interchain transfer is being forwarded, and the endpoint signature becomes incompatible with the signature prescribed in the interchain transfer's information.
- The ITS on the destination blockchain has lost its right to transfer the token since the interchain transfer was initiated.
- The amount to transfer to the destination address exceeds the maximum integer that can be represented in a smart contract of the destination blockchain. For example, on Sui and Stellar, integers are `u64`, while on EVMs, integers are `u256`.

R2: Users' incoming interchain transfers might never be successfully executed on Solana and they might never be refunded on the source blockchain.

For the same reasons described in R1 regarding outgoing interchain transfers, incoming interchain transfers might never be successfully executed.

Finally, unlike in R1, there might be a refunding mechanism implemented on the source blockchain, however if there is no such mechanism, then users would have lost their tokens if their incoming interchain transfers are never successfully executed on Solana.

Below are examples specific to Solana of incoming interchain transfers that systematically fail to be executed.

Example 1: A transfer fails if it makes a CPI resulting in a transaction size exceeding the limit of 1232 bytes (doc). In fact, interchain transfers by default require a lot of bytes for the accounts needed to call ITS and Gateway. Therefore, there is a stringent restriction on the number of bytes remaining for the accounts specifically needed by the destination program. The sender should be aware of this limit when building his interchain transfer, otherwise it might fail on Solana.

Example 2: Tokens are lost if the destination is a Token Account. Indeed, tokens are always transferred to the destination's Associated Token Account (ATA), however if the destination itself is a Token Account, then it has no way to manage tokens of its ATA.

Example 3: A transfer fails if the destination is an account that does not own its ATA anymore. Indeed, ITS sends tokens to the destination's ATA, however it fails if the ATA is not owned anymore by the destination. Note that this scenario is possible because for old ATAs, the original owner of the ATA was allowed to transfer the ATA's ownership.

R3: Users' registration of a token from Solana might not be successfully executed on the destination blockchain.

This is because, for reasons explained below, the registration of a token on a destination blockchain might either (1) never be executed, or (2) be incorrectly executed, or (3) systematically fail to be executed.

1) The token's registration might never be executed if for any reason the Axelar network does not forward it to the destination blockchain.

Example: The ITS Hub on the Axelar blockchain currently only forwards the 1st attempt to register a token to a given destination blockchain, and does not forward any subsequent attempts. Therefore, if the 1st registration fails for any

reason on the destination blockchain, then it would be impossible to attempt the registration again.

2) The token's registration might be incorrectly executed if the registration's information is altered by the Axelar network while being forwarded to the destination blockchain.

Example: The ITS smart contract on the destination blockchain might perform normalizations of the token's information (e.g. name and symbol), to comply with the blockchain's specific requirements.

3) The token's registration might systematically fail to be executed on the destination blockchain, i.e. even if it is possible to re-try the execution, it would never succeed.

Example: The registration's data (e.g. minter address, or decimals) is in a format incompatible with the destination blockchain.

R4: Users' registration of a token from a source blockchain might not be successfully executed on Solana.

For the same reasons described in R3 regarding outgoing token registrations, incoming token registrations from a source blockchain to Solana might never be successfully executed or correctly executed.

Example: The token deployer on the source chain has provided a symbol of length greater than 10 for his token, but on Solana it will be truncated to the first 10 characters. Indeed, the name and symbol of a token on Solana are truncated above the maximum allowed length (32 and 10, respectively).

Code Issues & Recommendations

C1: Tokens of inbound transfers with CPI are stuck and destination program can't spend them

Severity: Major **Status:** Fixed

Location

```
src/instructions/gmp/interchain_transfer.rs
    execute_interchain_transfer_handler
```

Description

Current behavior: In inbound token transfers with CPI, the tokens are sent to an ATA (link) whose authority is a program (link). Since programs can't sign but the Token Program requires the authority's signature for spending tokens, **the tokens are effectively stuck and the destination can't use them.**

Remark: If the program's deployer kept the program's private keys, then he could in principle unstuck the funds. However in practice:

- It might not be the case (e.g. the program was deployed by another program),
- Even if it is the case, it makes integrations cumbersome, as it forces the destination program to rely on the intervention of the program deployer (in particular, this forces a permissioned design).

Expected behavior: The destination program should be able to operate on the tokens it receives from inbound interchain transfers.

Worst consequence: ITS is practically unusable for interchain token transfers with CPIs.

Recommendation

We recommend, for inbound interchain transfers to a program, to send tokens to the ATA of a PDA associated with the destination program.

C2: Misleading parameter names

Severity: Minor **Status:** Fixed

Location

`src/state/user_roles.rs`
`contains`

Description

In the `UserRoles` struct, several methods use the parameter name `role` (singular) even though the `u8` parameter can represent multiple roles simultaneously as a bitmask. Specifically, `contains(role)`, `insert(new_role)`, and `remove(role)` all accept a `u8` that may encode multiple roles at once (e.g., `roles::MINTER | roles::OPERATOR`). This naming is misleading and does not convey that the parameter can represent a combination of roles.

Recommendation

We recommend renaming the parameters from `role` / `new_role` to `roles` / `new_roles` to accurately reflect that they can represent multiple roles.

C3: Unused error variants in RolesError

Severity: Minor **Status:** Fixed

Location

```
src/state/user_roles.rs
   RolesError
```

Description

Three error variants in `RolesError` are defined but never used: `ProposalMissingMinterRole`, `ProposalMissingOperatorRole`, and `ProposalMissingFlowLimiterRole`.

Recommendation

We recommend removing these unused error variants.