Audit Report

Axelar Network Audit

Ferbruary 6, 2022

Version 1.0

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by the Axelar Foundation to perform a security audit of the Axelar Cosmos to Ethereum bridging solution.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repositories:

https://github.com/axelarnetwork/solidity-cgp-gateway

Commit hash: `dab5a33db585df2fc6e1c20062b2811089d1d387f`

https://github.com/axelarnetwork/axelar-core

Commit hash: `120b011b37ae1f5228a5302065e7c67238be9160`

Audit Scope and Review Limitations

The Axelar codebase is still in active development. The scope of the current audit has been limited to verifying the security of asset transfers between Axelar native tokens and their ERC-20 representation on EVM compatible blockchains.

To this end, the audit has mainly focused on the EVM Cosmos SDK module and the related smart contracts.

Whilst additional modules have been reviewed, a full security audit of these modules has not been part of the process.

Due to active development on the codebase, future audits are recommended. In particular, the current limitations apply:

- **The threshold signature scheme used as an alternative to a more classic multisigmutlisig voting scheme is out of scope and still in development**
- **The current voting scheme assigns equal weight to all voters which may have consensus/incentive implications**

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.

2. Automated source code and dependency analysis.

3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:

    a. Race condition analysis

    b. Under-/overflow issues

    c. Key management vulnerabilities

4. Report preparation

Functionality Overview

The submitted protocol implements a cross-blockchain asset transfer solution between Cosmos and Ethereum. Assets (ERC-20 tokens) can be transferred from Ethereum to Cosmos and vice versa via a set of smart contracts controlled by a set of validators on a specific Cosmos SDK blockchain. Whilst the submitted codebase integrates further network and asset integrations, the scope of this audit was limited to Ethereum asset transfers.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged** or **Resolved**.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Setup function can be used by anyone to take control of the gateway contract	Critical	Resolved
2	Non-unique key for identifying voting topics implementations may lead to lost proposals	Major	Acknowledged
3	Gateway contract address depend on token symbol only and might clash	Major	Acknowledged
4	Expired polls can be voted on	Major	Acknowledged
5	Manual storage management in upgradability pattern used is error-prone and may lead to storage key conflicts	Minor	Acknowledged
6	ExportGenesis should return the genesis state instead of nil	Minor	Resolved
7	Panic used for non-eligible validators in inflation calculation breaks iteration for the other eligible bonded validator	Minor	Acknowledged
8	Single invalid key causes subsequent keys valid keys to be skipped when batch-registering external keys	Minor	Acknowledged
9	RouteIBCTransfer is not considering all the valid chain entries in case of error during iteration.	Minor	Resolved
10	Extreme value check is performed on inflation	Minor	Acknowledged
11	Lack of retrial limit for IBC transfers might cause validator funds to drain	Minor	Acknowledged

12	Access control modified with major side-effects introduces logic in eternal storage	Informational	Acknowledged
13	Excessive gas usage through burner contracts	Informational	Resolved
14	Solidity version pragma allows for versions with security vulnerabilities	Informational	Acknowledged

Project Risk Analysis

Audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

This section is meant to provide an indicator of the remaining risk. Users of the system should exercise caution.

Code Quality

In order to subjectively quantify the remaining risk, we provide a measure of the following code quality indicators: **code complexity**, **code readability**, **level of documentation** and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	**Status**	**Comment**
Code complexity	**High**	The codebase is very large and complex. Whilst the recommended Cosmos SDK design patterns are followed in most cases, the codebase lacks to maintain clear segregation between keepers and message servers, leading to execution flows that are hard to follow and verify.
Code readability and clarity	**Medium-low**	Readability is in line with other Cosmos SDK-based implementations. However, it is slightly compromised by the unusual flow distribution between keeper and message server components.
Level of Documentation	**Medium**	Informal documentation has been submitted for the audit. However, this has been found partially out of date and unsuitable for user-facing publication.

Test Coverage	Medium-low	Whilst some tests and mock implementations have been provided, the complexity of the protocol demands a more rigorous approach to testing combining unit testing with integration testing focusing on all error and edge cases.

Project Inherent Risk

In addition to code quality indicators, risk can be characterized by the nature of the project or protocol.

Criteria	Risk Level	Comment
Modularity Is the project a self-contained unit, such as a single smart contract, or does it have many moving parts?	High	The project relies on multiple components: A Cosmos SDK blockchain with several custom modules, a threshold cryptography library, and smart contracts on several external blockchains.
Technology Complexity Does the project rely on different technologies interacting with each other?	High	Yes, see above.
Degree of Experimentation Is the project implementing a well-known concept or does it implement experimental concepts, such as new economic models?	Medium	Bridges are fairly well understood, however the current design involves many moving parts, including threshold cryptography, upgradable smart contracts with complex addressing mechanisms and a reward mechanism for validators.
External Dependencies Does the project interact with external components, such as other protocols or oracles?	High	Axelar relies on several cryptographic protocol implementations, different EVM platforms and other external blockchains and a large number of Cosmos SDK modules.

Due to the medium to high risks involved with the bridge, we recommend the project to undergo ongoing security audits by multiple providers. We also recommend creating a treasury with an insurance fund for ongoing maintenance and compensation in case of any exploits. Additionally, insurance coverage could be acquired.

Detailed Findings

1. Setup function can be used by anyone to take control of the gateway contract

Severity: Critical

In the Solidity contract `contracts/src/AxelarGatewayMultisig.sol` the `setup` function can be called multiple times by anyone, allowing an adversary to take over control of the key privileged roles. This is due to the function being externally callable and not protected by any pre-condition other than not being callable on the implementation contract directly. However, since the storage contract is that of the proxy this guard does not prevent unauthorized invocations.

This issue also applies to `contracts/src/AxelarGatewaySinglesig.sol` (out of scope).

Recommendation

We recommend adding access control to this function.

Status: Resolved

2. Non-unique key for identifying voting topics implementations may lead to lost proposals

Severity: Major

In the Solidity contract `contracts/src/AdminMultisigBase.sol` function the hash of `msg.data` is used to identify proposal topics. This value is used as a key to store the topic in the eternal storage contract (key-value store).

However, `msg.data` simply encodes functions and parameters, meaning that repeated proposal of the same operation will result in the key being calculated with the same result. In this case, the previous proposal would be overwritten in storage and lost.

`msg.data` can also be malleable due to different libraries or wallets adding trailing zeros which could lead to key mismatch (see issue Manual storage management in upgradability pattern used is error-prone and may lead to storage key conflicts).

Recommendation

We recommend adding a unique parameter to the key used to store the topic.

Status: Acknowledged

The team states that in their use case proposal overwriting is no not problematic and that all used libraries encode `msg.data` consistently.

3. Gateway contract address depend on token symbol only and might clash

Severity: Major

Token contracts and burner contracts are deployed using the `create2` opcode, in order to deterministically calculate the addresses on both sides. Since the initialization code remains constant for all tokens the only distinguishing factor for different deployments is the salt used in the address calculation. To this end, the codebase uses the hash of the token symbol. Since the token symbol becomes the only identifier of an asset, tokens with the same symbol will clash, resulting in the same contract address. Effectively, only 3 bytes of input are used to produce a 32-byte hash value. This may lead to accidental overwriting of assets.

Recommendation

We recommend using values that provide a higher degree of entropy to generate the hash or keeping a registry of asset addresses shared between both sides of the bridge, instead of dynamically calculating the addresses.

Status: Acknowledged

4. Expired polls can be voted on

Severity: Major

In the `x/vote/types/types.go` module, the `Vote` method lacks a check whether the poll has been expired, allowing expired polls to be voted on.

Recommendation

We recommend checking the `Expired` state.

Status: Acknowledged

Team Reply: *"That is actually desired behaviour. We set the Expired flag after a configured amount of blocks to make it possible to retry a poll (otherwise we could get stuck while voting on deposits, tokens etc) if needed. But an expired poll is not automatically invalid, it just means it hasn't been completed after that configured amount of time. If late voters make the poll succeed after expiry that is not an issue."*

5. Manual storage management in upgradability pattern used is error-prone and may lead to storage key conflicts

Severity: Minor

The Solidity smart contracts use an eternal storage pattern using a hash of an identifier key as a manual storage pointer..

This introduces a source of potential errors since keys with the same name result in the same hash causing storage to be overwritten.

The reverse can also occur if the source of the key calculation may be encoded slightly differently by off-chain libraries. An example of this occurring is the `msg.data` field used as a key in `contracts/src/AdminMultisigBase.sol`. This field may have trailing zeros, depending on the wallet implementation used to interact with the smart contract, resulting in different key calculations and failure to look up the correct entry.

Recommendation

There are a number of alternative patterns for upgradability that provide eternal storage with upgradable logic. In addition, tooling exists automating some of the nuisances of memory management. An example of this is the OpenZeppelin upgradability framework (https://docs.openzeppelin.com/upgrades-plugins/1.x/)

Status: Acknowledged

6. ExportGenesis should return the genesis state instead of nil

Severity: Minor

In the `x/axelarnet` module, the genesis export returns `nil`.

Recommendation

We recommend returning the module's genesis state variable in the `ExportGenesis` method.

Status: Resolved

7. Panic used for non-eligible validators in inflation calculation breaks iteration for the other eligible bonded validator

Severity: Minor

Function `handleTssInflation` in `x/reward/abci.go` uses panic during the bonded validator iteration when a validator is found to be not eligible. This will skip the other eligible bonded validators.

Recommendation

We recommend skipping only the uneligible validator and continuing to handle other validators in the bonded validator list.

Status: Acknowledged

Team Reply: *"An error received from GetValidatorIllegibility is a panic-worthy scenario since a bonded validator should have a corresponding consensus address. This follows standard Cosmos practice of calling panic when an error is received from a function that is never expected to fail."*

8. Single invalid key causes subsequent valid keys to be skipped when batch-registering external keys

Severity: Minor

In the `RegisterExternalKeys` function in `x/tss/keeper/msg_server.go`, the iteration skips valid keys when an invalid key is present in the slice by returning from the method when it finds any validation error.

Recommendation

We recommend validating all the keys first and then only calling adding valid keys. Alternatively, the method may choose to panic and register no key at all.

Status: Acknowledged

Team Reply: *"This is not applicable since returning an error reverts to the original state in Cosmos, same as calling a panic, so none of the external keys will be set if one is invalid. To reply to Point 10, we expect a precise number of keys so we can't ignore invalid keys, and we think it's bad practice to process a message with any invalid input (this should never occur anyways from an honest user)."*

9. RouteIBCTransfer is not considering all the valid chain entries in case of error during iteration

Severity: Minor

In the `x/axelarnet` module message server function, the `RouteIBCTransfers` returns on encountering an error during iterations. This will cause the method to skip all other entries in the slice over which the iteration is performed.

Recommendation

We recommend using `continue` instead of `return`.

Status: Resolved

10. Extreme value check is performed on inflation

Severity: Minor

In the `validateExternalChainVotingInflationRate` and `validateTssRelativeInflationRate` functions of `x/reward/types/params.go` validation performed on the parameter for inflations are done at the extreme edge case of values 0 and 1.

However, when the value is set to 0, it will cause the reward to be calculated as 0, which is not in favour of accounts or validators who are actively participating in the protocol activity, therefore discouraging participation.

On the other extreme, a value of 1 might provide very large rewards, which also negatively affect the protocols economics.

Recommendation

We recommend incorporating min and max values and relevant validation for the inflation parameters to exclude extreme values.

Status: Acknowledged

Team Reply: *“`TssRelativeInflationRate` specifies the percentage of the Tendermint inflation rate that is added as a bonus, and thus is naturally capped by the current Tendermint inflation rate. In fact, we planned to set this to 100%, i.e 1.0 for the mainnet. And initially, we do want the ability to disable inflation rewards, i.e set it to 0. In general, long term this should be governable by the community, and so we haven’t added many restrictions.”*

11. Lack of retrial limit for IBC transfers might cause validator funds to drain

Severity: Minor

In `audit-axelar-core/x/axelarnet/module.go`, IBC message error handling is handled.

In case of negative acknowledgement or timeout, the code resolves to retry the same packet again and again without maintaining any maximum retrial counter.

Every time a relayer is relaying a packet to the destination chain, it has to pay transaction fees. The continuous resending attempts might, therefore, lead to significant funds drainage in the form of transaction fees.

Recommendation

We recommend implementing a maximum limit for retries.

Status: Acknowledged

Team Reply: *"On packet-timeout, it's guaranteed by IBC that it won't timeout again. We don't think the acknowledgement error case is concerning unless there is a bug in the IBC protocol itself with the refund logic. And we don't want to miss any packets for our protocol."*

12. Access control modified with major side-effects introduces logic in eternal storage

Severity: Informational

In `contracts/src/AdminMultisigBase.sol` the `onlyAdmin` modifier is used to ensure only an admin role can call certain methods. However, the modifier also modifies state and introduces a significant amount of logic.

This introduces two problems:

Firstly, it breaks with common conventions of access control modifier naming and introduces side-effects, including aborting in executing without reverting in some cases.

Secondly, the use of this modifier introduces significant logic into the eternal storage part of the contracts, breaking the upgradability paradigm used.

Recommendation

We recommend not implementing any state-changing logic in access control modifiers.

Status: Acknowledged

13. Excessive gas usage through burner contracts

Severity: Informational

On the Ethereum side assets are burnt by deploying a special-purpose burner contract that self-destructs. There seems to be no advantage to using this approach and the gas usage is much higher than commonly used ERC-20 extensions for burning tokens.

Recommendation

We recommend using a permissioned burn method as part of the ERC-20 implementation.

Status: Resolved

14. Solidity version pragma allows for versions with security vulnerabilities

Severity: Informational

The codebase allows any Solidity compiler version from 0.8 to less than 0.9. A number of important compiler bugs have recently been fixed. In particular, version 0.8.4 fixed important security issues.

Recommendation

Consider locking the version pragma to a specific compiler version, preferably 0.8.4 or above.

Status: Acknowledged