# Security Audit Report

Axelar: axelarnet

Authors: Ranadeep Biswas, Mirel Dalcekovic

Last revised 4 August, 2023

## Table of Contents

# Audit Overview

## The Project

Axelar Network is a decentralized interoperability network that enables the transfer of value and data across different blockchain ecosystems.
It achieves this by providing a secure and decentralized platform that allows developers to build and deploy applications that can interact with multiple blockchain networks without requiring significant changes to the underlying protocols. In essence, Axelar aims to remove the barriers that prevent different blockchain networks from communicating and interacting with each other, thus creating a more interconnected and efficient blockchain ecosystem.

Axelar Network offers two types of cross-chain services: native token transfers and general message passing for Ethereum Virtual Machine (EVM) chains, with or without tokens attached. Recently, Axelar Network added a new feature enabling communication with Cosmos chains using its Global Messaging Protocol (GMP), which went live on the mainnet on April 11th. With these new features, Axelar Network now provides the ability to communicate with both EVM and Cosmos chains, making it an even more versatile and useful platform for developers building decentralized applications across multiple blockchain ecosystems.

## Scope of this audit

The audit took place from March 23, 2023 through April 18, 2023 by Informal Systems by the following personnel:

- Ranadeep Biswas
- Mirel Dalcekovic

During the audit, we focused on analyzing the x/axelarnet module, which is a small component of the Axelar Network's complex system. The remainder of the system was largely considered a "black box" during the audit.

Our analysis centered on the Axelar team's stated priorities, namely their use of the Inter-Blockchain Communication (IBC) protocol and the General Message Passing (GMP) functionality for communication with Cosmos chains. The GMP protocol is a decentralized messaging protocol that underlies the Axelar Network's cross-chain communication and asset transfer capabilities, enabling the transfer of native tokens between different blockchain networks with different consensus mechanisms.

It's important to note that the Axelar Network contains off-chain components that play a critical role in cross-chain transfers, as they are responsible for triggering these transactions.

## Conducted work

The audit team started its work on March 24th.
The development team's representative gave us an initial overview of the codebase, followed by one more detailed walkthrough of specific parts in the following week.
The audit team engaged in reviewing the code of the axelarnet module as well as performing some e2e testing on axelar's public testnet.

## Timeline

- week 1: Onboarding to the project, walkthrough of GMP feature, system understanding and axelarnet module first iteration of code inspection with focus on IBC usage, translation of denominations.
- week 2: Walkthroughs of Native token transfer features, the audit team inspecting the GMP feature, Native token transfers and rate limits. Development team updated the commit hash audited to the v0.33.0 version why the code was once again overviewed over the diffs created.
- week 3: The audit team setting up of local axelar chain node and connecting it to the testnet, adaptations on the testnet by axelar team (placing rate limits, fees to be ther than zero ) familiarize with API, running e2e

tests - GMP focus on the fee logic, code inspection of evm translation logic and fee logic. Formulation of the work performed.

## Conclusions

In general, we found the codebase to be of high quality. The documentation on the Axelar site is well-written and covers the live features comprehensively. The code is well-structured and easy to follow, and the test suite includes both unit and integration tests.

However, we faced some difficulties in understanding the underlying assumptions and design of the system. It took us some time to figure out how the modules communicate with each other and how cross-chain transfer flows are initiated. It would be helpful to provide call stack diagrams, state transitions, and details on the read/writes to the appropriate module's stores to facilitate understanding for future developers.

As a confirmation of high quality code, we did not find any critical or high severity issues in the codebase. Most of the issues we encountered were related to aesthetics and suggestions for improving the current design. Overall, we are impressed with the quality of the codebase.

## Further Increasing Confidence

The audit was limited to manual code review and manual analysis. We acknowledge that the Axelar development team follows excellent development practices, and they have performed comprehensive end-to-end testing of new features on their testnet before going live.

To enhance confidence in the protocol design and implementation, as a next step, we recommend creating a detailed protocol-level specification that lists all the protocol's properties and assumptions behind them. This would aid in understanding the intended behavior of the Axelar network and identifying potential protocol-level issues.

# Audit Dashboard

## Target Summary

- **Type**: Specification and Implementation
- **Platform**: Golang
- **Artifacts**:
  - axelarnet module at commit hash 4755f9cb4d
  - axelarnet module at release tag v0.33.0
  - evm-cosmos examples at commit has 6d53a78b95, not under auditing but to provide context:
  - diff PR created for axelarnet changes during the audit, for easier overview

## Engagement Summary

- **Dates**: 23.03.2023 to 18.04.2023
- **Method**: Manual code review & protocol analysis
- **Employees Engaged**: 2

## Severity Summary

| **Finding Severity** | **#** |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 4 |
| Informational | 4 |
| **Total** | **8** |

## System overview

The Axelar network serves as a hub, linking different ecosystems together. Since it is a blockchain, it can do much more than bridges connecting blockchains, as all connected chains can communicate with each other seamlessly. The network also enables the generation of a one-time deposit address that can receive cross-chain funds from any wallet, giving users a centralized experience during exchanges with other open protocols and networks.

To achieve high safety and liveness, Axelar has introduced protocols such as the Cross Chain Gateway Protocol (CGP) and the Cross Chain Transfer Protocol (CTP). Additionally, the network has an app-level protocol stack that sits on top of routing protocols, like CCG and other routing technologies, that allows application developers to connect their dApps on any chain and perform cross-chain requests.

The Axelar network offers several advantages, including the ability for blockchain platform builders to easily plug in their blockchains to all other blockchain ecosystems with only a threshold account needing to be set up on the chain to connect to the network. For dApp builders, hosting their dApps anywhere, locking, unlocking, transferring assets, and communicating with applications on any other chain via CTP API is possible. Users can interact with all applications across the ecosystem directly from their wallets.

Axelar allows sending cross-chain token transfers and messages between any Cosmos or EVM chains, providing a uniform cross-chain experience for developers.

**Gateway contracts**: (Not under the scope of the audit.)
To execute cross-chain transactions, Gateway contracts are deployed on all EVM-connected chains. The Gateway contract is controlled by a key that is held jointly by all Axelar validators. To achieve this, a multi-party cryptography scheme is used, where the key is divided into many pieces, known as key shares. The number of shares is equal to the amount of staked AXL tokens held by the validator. For an action to be executed, the threshold must be reached.

Before transactions are authorized by the Gateway, they must first be confirmed by a validator vote. Once authorized, Axelar burns tokens or approves GMP transactions.

**Validators:** To participate in consensus, validators must verify all cross-chain activity processed by the Axelar network. This involves running nodes for Axelar-supported chains and observing external chains for activity.

The process involves a Delegated Proof of Stake (DPoS) mechanism where validators hold states of other connected chains and create an "incoming bridge." While signing details and threshold schemes are important for this mechanism, they are complex and were not analyzed in detail as they were outside the scope of the audit.

**Relaying services:** Can be performed by anyone, no form of trust is required - existing relayer service can be used or apps can build and use their own relayers.

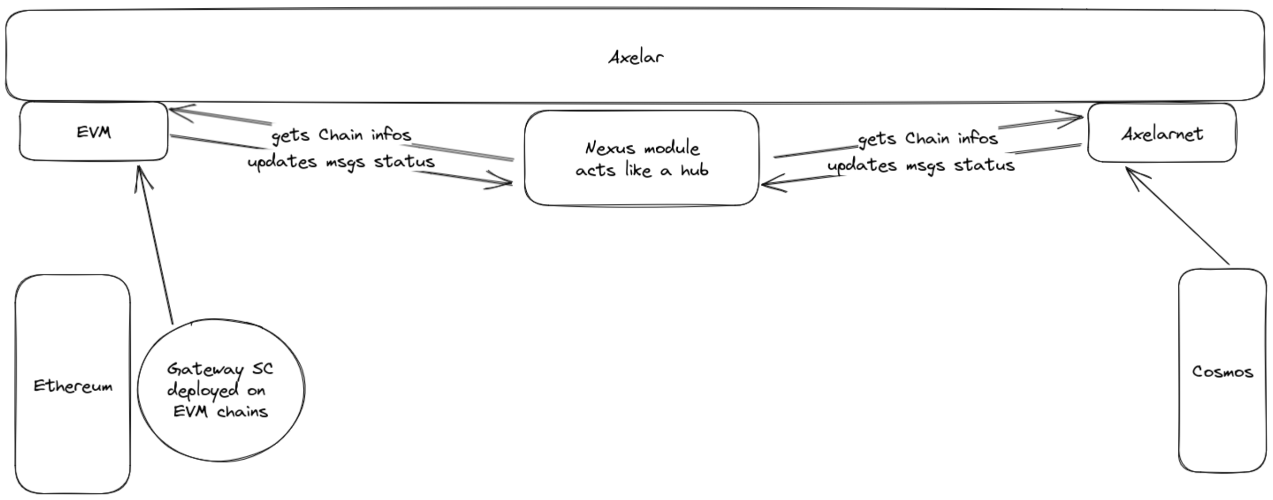
**Gas receiver**

Application developers using Axelar for General Message Passing have two options:

1. They can build their own relayer services on the destination chain, which will cover the gas fees required for the final executable smart contract call;
2. They can pay gas fees with the Axelar Gas Receiver on the source chain. Gas receiver is a smart contract that accepts tokens as payment to cover costs of contract execution for general message passing tx.
   Axelar relayer services will confirm the gas payment on the source chain, then automatically execute the smart contract call on the destination chain when it gets approved.

**Modules**

The axelarnet module serves as the gateway to the Cosmos side, while the evm module is responsible for the EVM chains. The Nexus module acts as a central hub for packet transfers between the chains. Although there are other modules, such as multisig/vote, which offer useful features like signing and voting, they are not relevant to the scope of this audit.



**Cross chain transfers**
Using the permissioned command in axelarnet, any IBC channel can be registered with an alias or chain name, which can then be used as a destination for token transfers and messages.
Two types of cross-chain sends are supported:

1. native token transfers and
2. general message passing (with or without tokens attached).

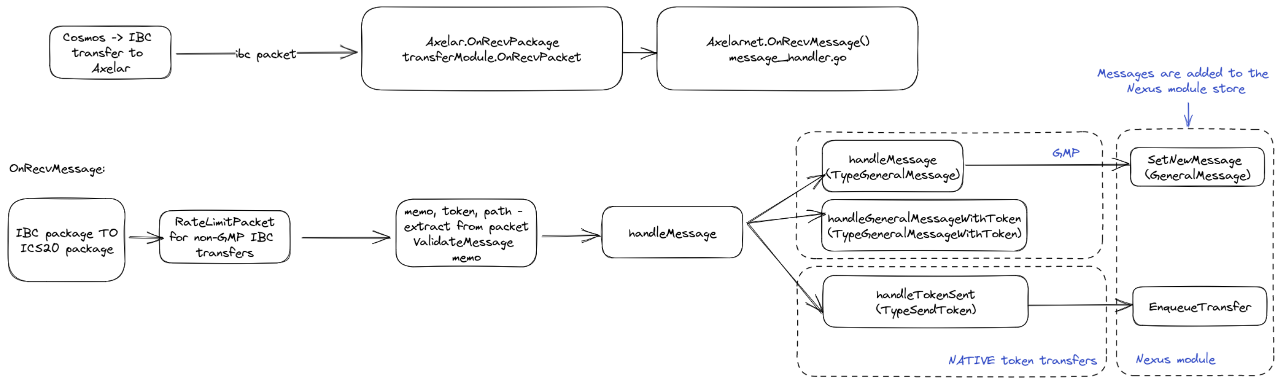
The scope of this audit is the cosmos side of axelar:

- axelarnet module
- and its use of the IBC.

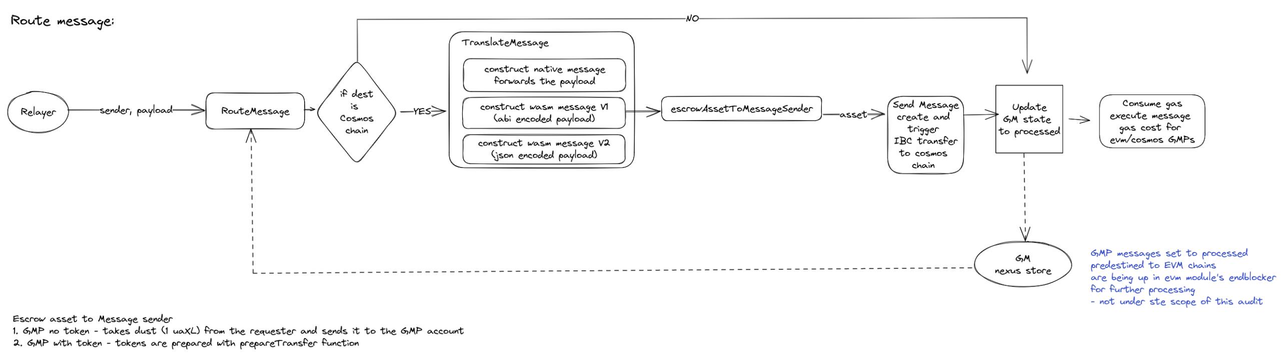
Our analysis will cover IBC rate limits, handling of failures in transfers, and denomination representations in axelarnet modules, including the translation of IBC denoms. We will also examine both types of cross-chain message flows through the axelarnet module.

## Cross chain message flows of importance for this audit are:

**Processing of cross chain messages received from other cosmos/evm chains:**


Cosmos -> IBC transfer to Axelar
ibc packet
Axelar.OnRecvPackage transferModule.OnRecvPacket
Axelarnet.OnRecvMessage() message_handler.go
Messages are added to the Nexus module store
OnRecvMessage:
IBC package TO ICS20 package
RateLimitPacket for non-GMP IBC transfers
memo, token, path - extract from packet ValidateMessage memo
handleMessage
handleMessage (TypeGeneralMessage)
GMP
handleGeneralMessageWithToken (TypeGeneralMessageWithToken)
handleTokenSent (TypeSendToken)
SetNewMessage (GeneralMessage)
EnqueueTransfer
NATIVE token transfers
Nexus module

**The execution flow for existing GMP messages**
Is triggered with `RouteMessage` function on the axelarent API.
The flow is as described on the diagram.


Route message:
Relayer
sender, payload
RouteMessage
if dest is Cosmos chain
YES
NO
TranslateMessage
construct native message forwards the payload
construct wasm message V1 (abi encoded payload)
construct wasm message V2 (json encoded payload)
escrowAssetToMessageSender
asset
Send Message create and trigger IBC transfer to cosmos chain
Update GM state to processed
Consume gas execute message gas cost for evm/cosmos GMPs
GM nexus store
GMP messages set to processed predestined to EVM chains are being up in evm module's endblocker for further processing - not under ste scope of this audit
Escrow asset to Message sender
1. GMP no token - takes dust (1 uaXL) from the requester and sends it to the GMP account
2. GMP with token - tokens are prepared with prepareTransfer function

Execution of cross chain messages is triggered from:

1. axelarnet module's end blocker - for native token transfers
2. with messages existing on axelarnet API:

- native/IBC token transfers:
    - RouteIBCTransfers - sets up everything for axelarnet's
    - ExecutePendingTransfers
    - RetryIBCTransfer
- GMPs:
    - RouteMessage - triggers the processing of the GMPs across the system

Given the complexity of the Axelar network and the presence of off-chain components, certain parts of the system could not be thoroughly analyzed within the time constraints of this project. It's worth noting that some components, such as the relayer services, are black-boxed and were not audited in this study.

However, the analysis carried out offers valuable insights into the functionality of the axelarnet module and its role in enabling cross-chain transactions in the network. To ensure a comprehensive and high-quality audit, we focused on understanding the essential components of the system.

The brief overview of the system provided in this report represents our collective understanding.

## Threat inspection

The following section outlines the three-week audit performed on the axelarnet module's code. Due to the system's complexity, as described in our system overview and internal notes, it was challenging to analyze all possible impacts and cases that could lead to issues related to or arising from the axelarnet module. We spent most of our onboarding time understanding and abstracting the components that were not audited and gaining insights into the system's behavior that impacts or is dependent on the module being audited.

The audit's scope was to analyze the functionality of the axelarnet module as a bridge on the Cosmos side of the Axelar network, whose primary role is to process and direct messages from the Axelar network to the destination chain. Since the module is dependent on many Axelar network components and communicates with them, our analysis was limited, and we considered most of these components as black boxes.

The audit was primarily focused on the scope discussed during the kick-off meeting:

- IBC usage
- IBC denoms translation
- General Message Passing
- EVM message translation on the Cosmos side

During the initial stage of the audit, the auditing team spent time identifying potential threats and analyzing the impact of the axelarnet module. Our primary objective was to validate the design's correctness and identify any issues in the codebase that could result in the realization of these threats. To achieve this, we held weekly sync meetings with the axelar team to discuss possible flaws in the system and gain insights into the current behavior of the system.
Through these efforts, we were able to validate that the identified threats could not be realized.

The following analysis contains our conclusions and the explanation of the axelarnet module's design on how those threats are mitigated.
In addition to our threat inspection analysis, we have documented our results as findings and executed end-to-end test cases.
We have shared our findings and testing results with axelar team representatives in the form of a list of minor issues and suggestions for design and implementation improvements.

It should be noted that, while we do our best in the analysis below, a security audit can by no means guarantee the absence of threats: the real guarantees can be provided only via systematic quality assurance, including manual and automated testing, and, ultimately, via the usage of formal methods, such as model-based testing or formal verification.

We have identified main categories of threats:

- **Halting of the axelar network chain**. As main IBC transferring logic is placed in the end blocker of the axelarnet module, we took the effort to inspect whether and how the logic or possible panics from deeper in the call stack could lead to halting the chain.
- **Possible exploit of funds** We analyzed manipulation with the deposit address, to see if there are certain design flaws that could lead to stealing the funds from this intermediatory account. Also, Fee logic was analyzed to confirm that the fee payment (if provided) is a prerequisite for placing the specific cross chain transfer in the waiting queue for execution. We questioned permissionless registration of chains and assets, with the possibility of placing the rate limit for a specific asset could be potentially dangerous, so we proved that those messages are executed only in case of the user holding the appropriate permission role.
- **dDoS attacks** - We analyzed the possibilities of the malicious or faulty creation of `Native token transfer` or `General Message Passing` messages. Also, since there are state transitions defined for both types of the cross chain messages, we were considering possible faulty state transitions for IBC transfer and GMPs in nexus and axelarnet modules. We were analyzing the possibility of creating cross chain messages that will fail to execute, and the possibility of them being triggered for execution each time, which could lead to denial of cross chain transfer services.

## Halting the axelar network chain

Halting of the chain is always possible with complicated begin or end blocker computations or with panics that could arise deep in the call stack.
As a potential place of issues in the axelarnet module, we detected the Endblocker logic.

Logic is used to process native cross-chain transfers as opposed to cross-chain messaging. There is a queue of cross chain transfers waiting to be sent per chain. The end blocker retrieves it and sends them out making sure any errors/panics when interacting with IBC revert the cached context, and allow us to continue processing other transfers.

RouteIBCTransfers execution is an entry point, since the IBCTransfers' are queued for the execution with this function, that is: all the CrossChainTransfers that are read from the nexus module (respecting tranferLimit number) are after this marked as pending in the nexus module's store and added to the axelarnet's KVStore queue with:

code

```
for _, p := range pendingTransfers {
            token, sender, err := prepareTransfer(ctx, s.Keeper, s.nexus, s.bank,
s.account, p.Asset)
            if err != nil {
                s.Logger(ctx).Error(fmt.Sprintf("failed to prepare transfer %s: %s",
p.String(), err))
                continue
            }


            funcs.MustNoErr(s.EnqueueIBCTransfer(ctx, types.NewIBCTransfer(sender,
p.Recipient.Address, token, portID, channelID, p.ID)))
            s.nexus.ArchivePendingTransfer(ctx, p)
        }
```

IBC Transfers are queued in the KVStore queue, under the special key

```
`ibcTransferQueueName = "route_transfer_queue"
```

The current design making it impossible to materialize the threat:

- There is a limit of the number of transfers processed per block to avoid being DOS'd and have a reasonable block gas limit: code

```
    endBlockerLimit := bk.GetEndBlockerLimit(ctx)
```

so that sending of the IBC transfers is limited to the maximum of the elements in the queue or with this endblocker limit.

- Cashed context is used in code
  making it possible to ignore the transfers failed due to panics, revert all the changes made to it and to proceed with the sending of the next IBC transfer.

```
_ = utils.RunCached(ctx, bk, func(cachedCtx sdk.Context) ([]abci.ValidatorUpdate,
error) {
            err := ibcKeeper.SendIBCTransfer(cachedCtx, transfer)
```

The failed transfers will not "pollute" the queue, so it is not possible to disable the endblocker triggering IBC transfer's logic, since:

- the IBC transfer is marked as `Archived` in the nexus module - which is making it impossible to again be found in the IBC tranfers end blocker queue,
- the queue is updated and the IBC transfer is removed as soon as the sending is triggered, so it can not remain in the queue for the next blocks end blocker,
- the axelarnet KVStore is updated for the Trasfer ID with the status `TransferFailed`.

In case of succesfully sending the IBC transfer from within the end blocker logic, te axelarnet module will be waiting for the ack to be received, which will once again updated the appropriate nexus module and axelarnet modules states.

## Possible exploit of funds

**Desposit address and messages analyzed:**

- deposit address is created with `Link` message
- permissionless `ConfirmDeposit` message

Flow of transferring the tokens via native token transfer flow:

To be able to transfer the funds from the axelar chain to some other chains' receiver - there is a mechanism of creating a deposit address used in the process of cross chain transferring of funds as a starting point. The deposit address is generated as hashed 32 bits containing destination chain info, recipient address on that chain and nonce of the Link Msg transaction, which is making it unique across the system.

```
// NewLinkedAddress creates a new address to make a deposit which can be transferred
to another blockchain
func NewLinkedAddress(ctx sdk.Context, chain nexus.ChainName, symbol, recipientAddr
string) sdk.AccAddress {
    nonce := utils.GetNonce(ctx.HeaderHash(), ctx.BlockGasMeter())
    hash := sha256.Sum256([]byte(fmt.Sprintf("%s_%s_%s_%x", chain, symbol,
recipientAddr, nonce)))
    return hash[:address.Len]
}
```

Even though the deposit address is unique, it is possible and expected to be reused if the users are sending tokens to the same recipient - since the address is not automatically deleted after the transfer is made.

After creation of the deposit address, the next step is to deposit the funds from the sender's account to the destination address, in order to make it possible for the system to trigger the IBC transfer. Once the funds are deposited, relayers should call the ConfirmDeposit function, placing the IBC transfer in the nexus module and making it possible for the network to process it. Even though it is expected that relayers are the ones triggering the process of the IBC transfer, the ConfirmDeposit function is completely permissionless, which makes it possible for:

- any user to trigger the ConfirmDeposit and in case of deposit existing on the address, the IBC transfer is created for funds to be transferred to the destination.
- In order for the transfer to be approved, fees should be paid - which means the deposit address should contain funds larger than the fee amount.
- The addresses can not be deleted intentionally or automatically after the transfer is triggered - the current implementation is actually prohibiting the malicious user from deleting the deposit address with just sending the minimum fee amount.

While we did not identify any issues with the deposit address and its usage that could lead to the exploitation of funds, we did observe an interesting behavior. Once a deposit has been made, it is not possible to withdraw the funds, and if the cross-chain transfer fails, the funds remain stuck in the escrow accounts. This has been identified as one of the findings of our audit: IF-AXELAR-05.md

**Fee logic**
Paid fees are a prerequisite for relayer's triggering the cross chain transfer or for an axelarnet module to place the cross chain message in the `Approved` (GMP) or `Pending` (IBC transfers) state.

**Critical chain management functions**
IT was confirmed that certain messages existing on the axelarnet API should be permissioned, due to being critical for the entire system.
Registering the assets, chains as well as placing limits for certain denominations are assumed to be valid and correct, so these messages are permissioned.
Here is the list of messages and expected permission roles:

- RegisterIBCPathRequest - ROLE_CHAIN_MANAGEMENT
- AddCosmosBasedChainRequest - ROLE_ACCESS_CONTROL
- RegisterAssetRequest - ROLE_CHAIN_MANAGEMENT
- RegisterFeeCollectorRequest - ROLE_ACCESS_CONTROL

The AnteHandler will fail if the signer of the message does not hold the appropriate role.

```
// AnteHandle fails if the signer is not authorized to send the transaction
func (d RestrictedTx) AnteHandle(ctx sdk.Context, tx sdk.Tx, simulate bool, next
sdk.AnteHandler) (sdk.Context, error) {
    msgs := tx.GetMsgs()
    for _, msg := range msgs {
        signer := msg.GetSigners()[0]
        signerRole := d.permission.GetRole(ctx, signer)

        switch permission.GetPermissionRole((msg).(descriptor.Message)) {
        case permission.ROLE_ACCESS_CONTROL:
            if permission.ROLE_ACCESS_CONTROL != signerRole {
                return ctx, sdkerrors.Wrapf(sdkerrors.ErrInvalidRequest, "%s is not
authorized to send transaction %T", signer, msg)
            }
        case permission.ROLE_CHAIN_MANAGEMENT:
            if permission.ROLE_CHAIN_MANAGEMENT != signerRole {
                return ctx, sdkerrors.Wrapf(sdkerrors.ErrInvalidRequest, "%s is not
authorized to send transaction %T", signer, msg)
            }
        default:
            continue
        }
    }

    return next(ctx, tx, simulate)
}
```

So, registering the invalid fee collector or placing the higher rate limit for a certain denomination is impossible by design.

## Distributed Denial of Service attacks

- We analyzed the possibilities of denial of service due to malicious use or faulty behavior of the axelarnet module.
  Since the axelar network:
- has offloaded the triggering of the cross chain transfers to the off chain components - relayers
- execution of the IBC transfers is limited to a certain number: `endblockerLimit` and `transferLimit`

- execution of the GMP message is triggered with `RouteMessage` message on axelarnet module
- does not trigger the cross chain transfer until the fees are paid
- the failed transfers are triggered with retrying the exact transfer with `RetryIBCTransfer`
- rate limits could be reached easily even for the failed or time out ibc transfers and this is an increased attack vector since the transfers could be blocked, even after resetting the rate limits. However, since the axelar network has permissioned mechanisms of fixing these situations, it is ok with the axelar team to have these potential issues and this is a result of a trade off: dragonberry like issues vs increased rate limit attack vector. This is explained in more detail in the IF-AXELAR-07.

We conducted a code inspection to identify any loops that may cause a denial of service (DoS) attack.

After conducting a thorough analysis of the design and code inspection, we did not identify any additional potential causes of a DoS attack.

## Findings

| Title | Type | Severity | Impact | Exploitability | Issue |
|---|---|---|---|---|---|
| IF-AXELAR-01: Various Minor Code Improvements | IMPLEMENTATION | 0 INFORMATIONAL | 0 NONE | 0 NONE | |
| IF-AXELAR-02: Event is not emitted for executed RouteMessage | IMPLEMENTATION | 0 INFORMATIONAL | 0 NONE | 0 NONE | |
| IF-AXELAR-06: IBC wrapping implementation improvement | IMPLEMENTATION | 0 INFORMATIONAL | 0 NONE | 0 NONE | |
| IF-AXELAR-08: Hardcoded denom issue on the devnets | IMPLEMENTATION | 0 INFORMATIONAL | 0 NONE | 0 NONE | |
| IF-AXELAR-03: Simplify Relayers setup using FeeGrant | PROTOCOL | 1 LOW | 1 LOW | 1 LOW | |
| IF-AXELAR-04: Additional queries over nexus module for better UX | IMPLEMENTATION | 1 LOW | 1 LOW | 1 LOW | |
| IF-AXELAR-05: Once the funds are deposited on the deposit address they can not be refunded | PROTOCOL | 1 LOW | 1 LOW | 1 LOW | |
| IF-AXELAR-07: Current rate limits design could be improved | IMPLEMENTATION | 1 LOW | 1 LOW | 1 LOW | |

## IF-AXELAR-01: Various Minor Code Improvements

| | |
|---|---|
| **Title** | IF-AXELAR-01: Various Minor Code Improvements |
| **Project** | Axelar: axelarnet |
| **Type** | **IMPLEMENTATION** |
| **Severity** | **0 INFORMATIONAL** |
| **Impact** | **0 NONE** |
| **Exploitability** | **0 NONE** |
| **Issue** | |

### Involved artifacts

- x/axelarnet/handler.go
- x/axelarnet/keeper/msg_server.go
- /x/axelarnet/types/msg_register_asset.go

### Description

Here is the list of aesthetic and minor code improvements and issues around logging found during the code inspection of the axelarnet module. They do not pose a security threat nor do they introduce an issue, but the following suggestions are shared to improve the code readability and logging.

1. RouteIBCTransfers(): logging of the executed handler, not correct code
2. GetChain() should panic in cases of system's input, due to the client's development practice? For x/axelarnet module analyzed:

- ExecutePendingTransfers(): this is coming from the system, not finding the expected chain registered in code should panic.
- RouteIBCTransfers(): chains are retrieved from the system, looping through them in code, in case when not found, should panic.

3. OnRecvMessage(): when the chain is not registered use the input string for logging the error, to provide info, in the code
4. ValidateBasic() checks for RegisterAssetRequest(): missing limit validation could be explained differently in code - since the limit does not need to be a positive number and this is a way of prohibiting the transfer of the tokens, adding a comment about the intended behavior of the features would be valuable.
5. RegisterAsset() calls registering asset on axelarnet as well, but could be already registered. It is nicer if we skip this call in the code in case of an already added asset to the axelarnet chain by just adding the if statement.

### Problem Scenarios

- There are no issues with the listed cases above, the logging may be misleading and those are improvements of the code, making it more readable and consistent across the axelarnet module.

## Recommendation

- As explained in the Description section.

## IF-AXELAR-02: Event is not emitted for executed RouteMessage

| | |
|---|---|
| **Title** | IF-AXELAR-02: Event is not emitted for executed RouteMessage |
| **Project** | Axelar: axelarnet |
| **Type** | **IMPLEMENTATION** |
| **Severity** | **0 INFORMATIONAL** |
| **Impact** | **0 NONE** |
| **Exploitability** | **0 NONE** |
| **Issue** | |

### Involved artifacts

- x/axelarnet/keeper/msg_server.go
- proto/axelar/axelarnet/v1beta1/events.proto

### Description

RouteMessage API function is missing an event being emitted at the end of the execution.
New type of event `GMPMessageProcessed` (or similar) should be defined in axelarnet.proto in order to emit the appropriate event for updating the GMP status in the nexus store.

### Problem Scenarios

Since a lot of features in an axelar network system are depending on the emitted events (relayers are notified with cross chain status changes and fees being paid) there could be a possible impact.
Currently, it seems that no components in the system are expecting this event, so it should be added to keep consistency.

### Recommendation

Defining and emitting the mentioned event, as described.

## IF-AXELAR-03: Simplify Relayers setup using FeeGrant

| | |
|---|---|
| **Title** | IF-AXELAR-03: Simplify Relayers setup using FeeGrant |
| **Project** | Axelar: axelarnet |
| **Type** | PROTOCOL |
| **Severity** | 1 LOW |
| **Impact** | 1 LOW |
| **Exploitability** | 1 LOW |
| **Issue** | |

### Involved artifacts

- x/axelarnet/keeper/msg_server.go
- x/axelarnet/client/cli/tx.go
- Relayer setup

### Description

Currently there are certain transaction fees that are paid by the sender of the tx (for messages: CallContract) while the idea is for relayers to pay for all the fees needed
to successfully transfer and execute the cross chain message on the destination chain.

### Problem Scenarios

The presented finding is actually an idea and a suggestion for improving the system's design making the axelar network consistent with all the GMPs created, both from axelar chain and evm/other cosmos chains.

### Recommendation

The suggested approach is only an idea, which must be further elaborated and better thought through.
There could be a solution for this issue by using the authorization module and granting some other account to pay for the fees of executing the transaction.
One interesting approach to resolve this could be with using the fee granter. This is not the most natural use of this feature, but we feel there is some space for getting exactly what is needed in order to resolve the complicated relayer setup needed to surpass this issue.

For CallContract function UseGrantedFees could be used if fee granter is defined for context for which the execution of the message is called at client context

```
// UseGrantedFees will try to pay the given fee from the granter's account as
requested by the grantee
```

```
func (k Keeper) UseGrantedFees(ctx sdk.Context, granter, grantee sdk.AccAddress, fee
sdk.Coins, msgs []sdk.Msg) error {
    f, err := k.getGrant(ctx, granter, grantee)
    if err != nil {
        return err
    }

    grant, err := f.GetGrant()
    if err != nil {
        return err
    }

    remove, err := grant.Accept(ctx, fee, msgs)

    if remove {
        // Ignoring the `revokeFeeAllowance` error, because the user has enough
grants to perform this transaction.
        k.revokeAllowance(ctx, granter, grantee)
        if err != nil {
            return err
        }

        emitUseGrantEvent(ctx, granter.String(), grantee.String())

        return nil
    }

    if err != nil {
        return err
    }

    emitUseGrantEvent(ctx, granter.String(), grantee.String())

    // if fee allowance is accepted, store the updated state of the allowance
    return k.GrantAllowance(ctx, granter, grantee, grant)
}
```

DeductFeeDecorator example:

```
deductFeesFrom := feePayer

    // if feegranter set deduct fee from feegranter account.
    // this works with only when feegrant enabled.
    if feeGranter != nil {
        if dfd.feegrantKeeper == nil {
            return ctx, sdkerrors.Wrap(sdkerrors.ErrInvalidRequest, "fee grants are
not enabled")
        } else if !feeGranter.Equals(feePayer) {
            err := dfd.feegrantKeeper.UseGrantedFees(ctx, feeGranter, feePayer, fee,
tx.GetMsgs())
            if err != nil {
                return ctx, sdkerrors.Wrapf(err, "%s not allowed to pay fees from %s",
 feeGranter, feePayer)
            }
        }
        deductFeesFrom = feeGranter
    }
```

## IF-AXELAR-04: Additional queries over nexus module for better UX

| | |
|---|---|
| **Title** | IF-AXELAR-04: Additional queries over nexus module for better UX |
| **Project** | Axelar: axelarnet |
| **Type** | IMPLEMENTATION |
| **Severity** | 1 LOW |
| **Impact** | 1 LOW |
| **Exploitability** | 1 LOW |
| **Issue** | |

### Involved artifacts

- /x/nexus/client/cli/query.go

### Description

The only way to filter out the needed GMP message Id for further processing when the actual user - not the off chain components (relayers) are triggering the execution is really hard.
It consists of manually filtering the events from the transactions executed.

It would be very useful and considered a UX improvement to add the possibility to query the pending GMPs over the nexus module.

As additional feedback from a user new to the system: The Axelarnet and Nexus CLI are very similar. Almost all transactions happen via axelarnet, but then queries are at the nexus subcommand. This can be confusing for users.

### Problem Scenarios

As explained, the UX could be significantly improved with additional queries.

### Recommendation

Even though the expected flow is for relayer to keep track of the needed message IDs, these queries would be valuable for testing purposes, so we suggest adding:

- paginated query that would list all the pending GMP cross chain messages
- filtered query for listing all the GMP message IDs for a specific sender (chain, address) and receiver (chain, address) and payload hash.

## IF-AXELAR-05: Once the funds are deposited on the deposit address they can not be refunded

| | |
|---|---|
| **Title** | IF-AXELAR-05: Once the funds are deposited on the deposit address they can not be refunded |
| **Project** | Axelar: axelarnet |
| **Type** | PROTOCOL |
| **Severity** | 1 LOW |
| **Impact** | 1 LOW |
| **Exploitability** | 1 LOW |
| **Issue** | |

### Involved artifacts

- x/axelarnet/keeper/msg_server.go

### Description

Once the user deposits a certain amount on the linked deposit address, it is not possible for a user to change their mind and withdraw the deposited funds back, giving up the intended cross chain transfer of the funds.
Once the ConfirmDeposit function is called, the funds will be sent to the defined destination address.
Even in the case of a failure on the destination chain, when the deposited amount will be locked in the escrow accounts - there seems to be no way to refund your assets.

### Problem Scenarios

The auditing team is of the opinion that this feature could be of use to the axelar network users, since now once the deposit is made - the transfer will be done as soon there is enough balance to cover the fees.

### Recommendation

Consider adding new feature WithdrawDeposit() that would enable users to withdraw their deposits prior to the relayer confirming the deposit and triggering the cross chain transfer.
One idea is to keep track of each user and the deposit amount (create records in the store), which could be deleted once the ibc transfer is confirmed.

### Agreements regarding the severity

Agreements regarding the severity
This finding is something that the axelar team is aware of and this is intentionally missing from design.
EVM side has this feature covered with a smart contract, it is not a part of the protocol level.

Since this feature could be added on the application level and axelar team does not consider this important enough (since it is a consequence of the user error) to place it in the protocol level, we agreed to downgrade the severity.

The main goal was to keep the protocol as simple as possible.

Axelar team considers this as a Low level finding.
Because of the complexity of the cross chain systems, funds could get stuck regardless of this missing feature, and from the axelar team's experience, this was mostly needed on the EVM side and was implemented on the app level.
The severity was updated to Low. Since this is the consequence of the user's mistake - we downgraded the exploitability to get the appropriate severity score.

## IF-AXELAR-06: IBC wrapping implementation improvement

| | |
|---|---|
| **Title** | IF-AXELAR-06: IBC wrapping implementation improvement |
| **Project** | Axelar: axelarnet |
| **Type** | IMPLEMENTATION |
| **Severity** | 0 INFORMATIONAL |
| **Impact** | 0 NONE |
| **Exploitability** | 0 NONE |
| **Issue** | |

### Involved artifacts

- x/axelarnet/module.go

### Description

IBC wrapping logic is implemented in module.go, and for `OnRecvPacket` we have an additional inside function `OnRecvMessage` call.
The `OnAcknowledgementPacket` wrapper also has additional logic executing after base function call.

### Problem Scenarios

-

### Recommendation

To maintain consistency, our suggestion is to move `OnAcknowledgementPacket` custom logic into a separate function, called `OnAcknowledgementMessage` .

## IF-AXELAR-07: Current rate limits design could be improved

| | |
|---|---|
| **Title** | IF-AXELAR-07: Current rate limits design could be improved |
| **Project** | Axelar: axelarnet |
| **Type** | IMPLEMENTATION |
| **Severity** | 1 LOW |
| **Impact** | 1 LOW |
| **Exploitability** | 1 LOW |
| **Issue** | |

### Involved artifacts

- rate limits feature design
- x/axelarnet/rate_limit.go
- x/nexus/keeper/rate_limit.go

### Description

With current design and implementation, rate limiting of the outgoing amount is performed in `SendPacket` with increasing> the outgoing rate limit's amount, but in the case of IBC transfer failure or packet timeout this amount will not be reduced.

The outgoing amount will be considered as successfully transferred to the other chain and the increased amount will remain as valid outgoing rate limit for a certain period of time - transferring epoch, defined with the block time and rate limit window with setting the rate limit for the system - SetRateLimit .

The failure of the IBC transfer will however impact the incoming rate limit amounts when processing:

- the received acknowledgement in `OnAcknowledgementPacket` , with increasing the incoming rate limit's amount:
- the packet timeout in `OnTimeoutPacket` , with increasing the incoming rate limit's amount.

This way we are limiting the outgoing and the incoming amount of tokens to the axelar chain for a certain period of time.

### Problem Scenarios

The auditing team discussed this with the axelar team on our sync meetings. We are aware that this is by design and works as expected.

We think that in cases of IBC transfers failure or packet timeouts, rate limits should not be changed.
With constant failures, we could get the situation that rate limits are reached for the outcoming or incoming

directions and further IBC transfers are blocked.
This will impact the system, and the user's will not be able to transfer the funds, at least for the epoch transferring time.

However, nothing is stopping the user, to hit the rate limits over and over again and prohibit the IBC transfers of the denomination.

On the other hand, if we consider the approach of reducing the outgoing limiting amounts we could have a situation that IBC transfers could exceed the rate limit amount (in case of some IBC internal issue) axelar chain could receive the information that the packet failed or timed out by mistake (but the transfer was actually successful).

## Recommendation

We feel that there is a place for improvement of this feature, even though we are not sure what the right approach would be.
We suppose this current approach is less risky, but it also has the DoS characteristics - since the IBC transfers could easily be blocked during the transferring epoch.

## Agreements regarding the severity

Since rate limits are designed and implemented to keep the system safe from bugs similar to the "dragonberry" bug, axelar team is aware of the increased attack vector with the currently selected approach. Decreasing the rate limits for outgoing direction could introduce less convenient issues that could not be easily fixed by the axelar network team's intervention.

Setting the rate limits is permissioned, so there is a mechanism to quickly resolve the issue (not having to wait for a governance proposal to be voted).
It has been agreed that this finding could be considered as a Medium severity one, due to the explanations provided.

In addition, the team representatives explained how they configured the system to provide extra security, which made it harder to exploit this attack vector:

- Rate limits are set only for assets with significant total value locked (TVL), so axelar avoids DoS issues for assets with small cap assets.
- Rate limits can be removed for any chain and asset combination, if needed.
- DoS issue via cycling between chains is even less possible if we take the time needed for a IBC roundtrip (~30s) and the timing for EVM chains, which is even longer.
- Rate limits are reset after 6 hrs.
- Axelar network has monitoring in place to detect if a certain fraction of the rate limit has been crossed to react if needed.

As a result, both the Informal Systems Audit team and the Axelar team representatives agreed to classify this finding as having a low risk of exploitation.

## IF-AXELAR-08: Hardcoded denom issue on the devnets

| | |
|---|---|
| **Title** | IF-AXELAR-08: Hardcoded denom issue on the devnets |
| **Project** | Axelar: axelarnet |
| **Type** | IMPLEMENTATION |
| **Severity** | 0 INFORMATIONAL |
| **Impact** | 0 NONE |
| **Exploitability** | 0 NONE |
| **Issue** | |

### Involved artifacts

- x/axelarnet/keeper/msg_server.go

### Description

During the e2e testing performed on the axelar devnet (devnet-ibc - without `uaxl` denom) there was an issue revealed with the `RouteMessage` execution.

When doing call contract to a cosmos chain, because it makes use of an IBC transfer and since the user themselves didn't provide a token (the relayer triggering RouteMessage attaches a dust token with using the hardcoded denomination with the code.

This will probably never happen on mainnet, as the native token is literally `uxal` (the hardcoded value). However, if this were to change, it could cause problems.

### Problem Scenarios

This is possible to encounter on the devnets with no initial supply of the `uaxl` tokens.
Commands to reproduce the error and the error received:

```
$ axelard tx axelarnet route-message
0xb3f3884d66f393c2be4de2c2d2f9d9c2dcf48486221b4a4e09db335961d7f5ea-1

000000027b227365745f6d657373616765223a207b20226d657373616765223a202248656c6c6f2049424
322207d7d --gas-prices 0.00005uibc --gas 200000 --from dev
...
"raw_log":"failed to execute message; message index: 0: 0uaxl is smaller than 1uaxl:
insufficient funds: axelarnet error"
```

## Recommendation

Here we are assuming that the denomination of the native asset is `uaxl`. While this is fine on testnet/mainnet, the assumption is incorrect on our devnets where the native denom is not `uaxl`. To fix this, you should either make the denomination configurable in RouteMessage or update it dynamically to match the native denomination.