Security Assessment of Utilizing Axelar Governance for EVM Smart Contract Upgrades

Axelar Inc
Version 1.0 – June 30, 2023

Prepared By
Parnian Alimi
Aleksandar Kircanski

Prepared For
Milap Sheth
Sergey Gorbunov

1 Executive Summary

Synopsis

During June 2023, Axelar engaged NCC Group to conduct a security review of the governance mechanism and the batched gateway transactions RPC call. One consultant was tasked for 8 person-days altogether in order to look at the PRs that implement these features.

The first item in scope, utilized Cosmos SDK's governance module and Axelar's General Message Passing (GMP) protocol to execute proposals that are approved by Axelar's on-chain governance. One application for these proposals, is to upgrade smart contracts (e.g. the Axelar Gateway contract) via Axelar governance. With this change, the token transfer rate limits can be controlled by multisig accounts as well as governance. The caveat of using the governance is its built-in time lock imposed delays. As such Axelar's app-level services have the option of being managed by the governance as well as separate multisig accounts (for timely responses to incidents).

The second item in scope, utilized the Ethereum JSON-RPC's support for batched requests to confirm multiple gateway transactions in an EVM chain. Deploying this change can decrease the RPC calls' volume and increase efficiency across the platform.

During this collaborative engagement, the Axelar team provided helpful documents and answered consultants' questions in order to speed up aspects of the review. NCC Group provided a best-effort review during a time-boxed engagement.

Scope

NCC Group's evaluation included:

1. **EVM Smart Contract Upgrade Controlled by Governance Mechanism**:
    - Axelar Core:
        - https://github.com/axelarnetwork/axelar-core/pull/1954
        - https://github.com/axelarnetwork/axelar-core/pull/1952
        - https://github.com/axelarnetwork/axelar-core/pull/1955
        - https://github.com/axelarnetwork/axelar-core/pull/1956
        - https://github.com/axelarnetwork/axelar-core/pull/1958
    - Axelar CGP Solidity
        - https://github.com/axelarnetwork/axelar-cgp-solidity/compare/main...feat/service-governance: commits between `b1e6e1f` and `78e6cbb` on `feat/service-governance` branch
    - Axelar GMP SDK Solidity
        - https://github.com/axelarnetwork/axelar-gmp-sdk-solidity/pull/40
2. **Batched RPC Calls Feature**:
    - https://github.com/axelarnetwork/axelar-core/pull/1929
    - https://github.com/axelarnetwork/axelar-core/pull/1939

Key Findings

The assessment uncovered the following findings:

- **After Proposal Deposit Hook Prematurely Rejects Proposals** which could lead to unexpected behavior
- **Inconsistent Minimum Deposit Validation** could lead to uncaught errors

Strategic Recommendations

NCC Group observed sparse documentation in some areas of the project. It is highly recommended to document the design choices as the project develops to ensure adequate knowledge transfer to future maintainers and auditors.

2 Dashboard

Finding Breakdown

Critical issues	0	
High issues	0	
Medium issues	0	
Low issues	2	🟨🟨
Informational issues	1	⬜
Total issues	**3**	

Category Breakdown

Data Validation	2	🟨⬜
Security Improvement Opportunity	1	🟨

🟪 Critical 🟥 High 🟧 Medium 🟨 Low ⬜ Informational

3 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
After Proposal Deposit Hook Prematurely Rejects Proposals	New	FF3	Low
Inconsistent Minimum Deposit Validation	New	RTB	Low
Insufficient Contract Address Validation for Call Contract Requests	New	F7Q	Info

4 Finding Details

Low

After Proposal Deposit Hook Prematurely Rejects Proposals

Overall Risk	Low	**Finding ID**	NCC-E005196-FF3
Impact	Undetermined	**Component**	axelar-core
Exploitability	Undetermined	**Category**	Security Improvement Opportunity
		Status	New

Impact

Axelar core's proposal deposit handling is more restrictive than Cosmos SDK flow, which could lead to unexpected behavior.

Description

Axelar core utilizes Cosmos SDK's governance API to handle the call contract proposal type. A proposal passes through 3 stages to be executed: submission, voting, execution. The Cosmos SDK documentation indicates that:

> When a proposal is submitted, it has to be accompanied with a deposit that must be strictly positive, but can be inferior to MinDeposit. The submitter doesn't need to pay for the entire deposit on their own. The newly created proposal is stored in an inactive proposal queue and stays there until its deposit passes the MinDeposit.

The Cosmos SDK's governance keeper handles a depositor's deposit in `AddDeposit`. It first validates that the proposal exists and is still depositable (it is in deposit or voting stage). Then it transfers the deposit amount to the governance module's account coins pool, and also adds the new deposit amount to the proposal's total deposit. Besides, the keeper adds/records the deposited amount for the depositor. As a result, the proposal depositors can call this API multiple times, to increase the proposal's total deposits until it reaches the proposal's required deposit.

Then the governance keeper calls Axelar's `AfterProposalDeposit()` hook, which will panic if the proposal has not reached its required deposit amount for every single contract call:

```go
// AfterProposalDeposit implements govtypes.GovHooks.
func (h Hooks) AfterProposalDeposit(ctx sdk.Context, proposalID uint64, _ sdk.AccAddress) {
  proposal := funcs.MustOk(h.gov.GetProposal(ctx, proposalID))

  switch c := proposal.GetContent().(type) {
  case *types.CallContractsProposal:
    minDepositsMap := h.k.GetParams(ctx).CallContractsProposalMinDeposits.ToMap()

    for _, contractCall := range c.ContractCalls {
      minDeposit := minDepositsMap.Get(contractCall.Chain, contractCall.ContractAddress)
      if !proposal.TotalDeposit.IsAllGTE(minDeposit) {
        panic(fmt.Errorf("proposal %d does not have enough deposits for calling contract %s on
        ↳ chain %s (required: %s, provided: %s)",
         proposalID, contractCall.ContractAddress, contractCall.Chain, minDeposit.String(), pro
         ↳ posal.TotalDeposit.String()))
      }
    }
  default:
    return
  }
}
```

Note that the `AddDeposit()` is called when the proposal is first submitted and depositing is allowed before the voting period begins. As such, the above check might reject the proposal too early.

Recommendation

Consider whether moving the minimum required contract call deposit check to the `AfterProposalVote()` hook is more appropriate. If expecting the total deposit limit to be fulfilled at the proposal submission is the desired behavior, document this design choice.

Location

https://github.com/axelarnetwork/axelar-core/blob/6805723fe25808946c4fdf0061964677 6bcebf92/x/axelarnet/keeper/hooks.go#L37

Low

Inconsistent Minimum Deposit Validation

Overall Risk	Low	**Finding ID**	NCC-E005196-RTB
Impact	Undetermined	**Component**	axelar-core
Exploitability	Undetermined	**Category**	Data Validation
		Status	New

Impact

Failure to consistently reject or accept an unset/empty minimum deposit leads to uncaught errors.

Description

The `CallContractProposalMinDeposits` type's basic validation rejects an empty `Coins` for a chain and contract address combination:

```
// ValidateBasic returns an error if the type is invalid
func (minDeposits CallContractProposalMinDeposits) ValidateBasic() error {
  chainContractAddressPairs := make(map[string]struct{})

  for _, minDeposit := range minDeposits {
    if err := minDeposit.ValidateBasic(); err != nil {
      return err
    }
```

The highlighted line above calls the `CallContractProposalMinDeposit` type's basic validator, which rejects an empty minimum deposit:

```
// ValidateBasic returns an error if the type is invalid
func (minDeposit CallContractProposalMinDeposit) ValidateBasic() error {
  if err := minDeposit.Chain.Validate(); err != nil {
    return err
  }

  if err := utils.ValidateString(minDeposit.ContractAddress); err != nil {
    return err
  }

  if minDeposit.MinDeposits.Empty() {
    return fmt.Errorf("min deposit cannot be empty")
  }

  if err := minDeposit.MinDeposits.Validate(); err != nil {
    return err
  }

  return nil
}
```

However, the `AfterProposalDeposit()` hook will simply skip missing minimum deposits for a chain and contract address combination, if it is not included in the `CallContractsProposalMinDeposits` map:

```go
27 // AfterProposalDeposit implements govtypes.GovHooks.
28 func (h Hooks) AfterProposalDeposit(ctx sdk.Context, proposalID uint64, _ sdk.AccAddress) {
29   proposal := funcs.MustOk(h.gov.GetProposal(ctx, proposalID))
30 
31   switch c := proposal.GetContent().(type) {
32   case *types.CallContractsProposal:
33     minDepositsMap := h.k.GetParams(ctx).CallContractsProposalMinDeposits.ToMap()
34 
35     for _, contractCall := range c.ContractCalls {
36       minDeposit := minDepositsMap.Get(contractCall.Chain, contractCall.ContractAddress)
37       if !proposal.TotalDeposit.IsAllGTE(minDeposit) {
38         panic(fmt.Errorf("proposal %d does not have enough deposits for calling contract %s
           ↳ on chain %s (required: %s, provided: %s)",
39           proposalID, contractCall.ContractAddress, contractCall.Chain, minDeposit.String(),
            ↳ proposal.TotalDeposit.String()))
40       }
41     }
42   default:
43     return
44   }
45 }
```

The getter on line 36 returns an empty `Coins` object, if the entry does not exist:

```go
// Get returns the minimum deposit for the given chain and contract address
func (m callContractProposalMinDepositsMap) Get(chain nexus.ChainName, contractAddress string)
↳ sdk.Coins {
  c := strings.ToLower(chain.String())
  address := strings.ToLower(contractAddress)

  if _, ok := m[c]; !ok {
    return sdk.Coins{}
  }

  return m[c][address]
}
```

Then the comparison on line 37 will return true since `coinsB` has a 0 length:

```go
// IsAllGTE returns false if for any denom in coinsB,
// the denom is present at a smaller amount in coins;
// else returns true.
func (coins Coins) IsAllGTE(coinsB Coins) bool {
  if len(coinsB) == 0 {
    return true
  }

  if len(coins) == 0 {
    return false
  }

  for _, coinB := range coinsB {
    if coinB.Amount.GT(coins.AmountOf(coinB.Denom)) {
      return false
    }
  }

  return true
}
```

Thereby, the `AfterProposalDeposit()` skips validating contract calls for which there are no minimum deposit set.

Recommendation

If ignoring validation for chain and contract address combinations that do not have a minimum deposit set is not the intended behavior, return an error in these cases.

Location

https://github.com/axelarnetwork/axelar-core/blob/6805723fe25808946c4fdf0061964677 6bcebf92/x/axelarnet/keeper/hooks.go#L27

Info

Insufficient Contract Address Validation for Call Contract Requests

Overall Risk	Informational	**Finding ID**	NCC-E005196-F7Q
Impact	Low	**Component**	axelar-core
Exploitability	Low	**Category**	Data Validation
		Status	New

Impact

Failure to reject invalid call contract requests early leads to unnecessary code execution.

Description

When a call contract command is received by an Axelar CLI client, it performs basic validations on the request:

```
34  // ValidateBasic executes a stateless message validation
35  func (m CallContractRequest) ValidateBasic() error {
36    if err := sdk.VerifyAddressFormat(m.Sender); err != nil {
37      return sdkerrors.Wrap(sdkerrors.ErrInvalidAddress, sdkerrors.Wrap(err,
      ↳ "sender").Error())
38    }
39
40    if err := m.Chain.Validate(); err != nil {
41      return sdkerrors.Wrap(err, "invalid chain")
42    }
43
44    if len(m.ContractAddress) == 0 {
45      return fmt.Errorf("contract address empty")
46    }
47
48    if m.Fee != nil {
49      if err := m.Fee.ValidateBasic(); err != nil {
50        return sdkerrors.Wrap(err, "fee")
51      }
52    }
53
54    return nil
55  }
```

The contract address validation on line 44, only ensures that the contract address parameter is not empty, however, it is also recommended to ensure that it is entirely composed of UTF8 characters, it is normalized as NFKC, and does not contain forbidden Unicode characters.

This invalid request will eventually be caught by the axelarnet's message server, when it processes the request which will validate it according to the destination chain's address format. As such the severity of this finding is set to informational.

Recommendation

Consider using the ValidateString() API from the `axelar-core` utils instead.

Location

https://github.com/axelarnetwork/axelar-core/blob/6805723fe25808946c4fdf0061964677
6bcebf92/x/axelarnet/types/msg_call_contract.go#L44

5 Implementation Review Notes

The following observation was not considered to have a security impact on the application and is left as a note.

Axelar-GMP-SDK-Solidity

- The TimeLock contract's `getTimeLock()` API silently returns *0* when the `hash` does not exist in contract's storage. However, if the caller attempts to `_finalizeTimeLock()` and the `hash` does not exist, the call will be reverted. These APIs' behavior is inconsistent. `getTimeLock()` was unused at the time of the audit.

6 Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Rating	Description
Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.
Medium	

Rating	Description
	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Category Name	Description
Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

7 Contact Info

The team from NCC Group has the following primary members:

- Parnian Alimi – Consultant
  parnian.alimi@nccgroup.com
- Aleksandar Kircanski – Consultant
  aleksandar.kircanski@nccgroup.com

The team from Axelar Inc has the following primary member:

- Milap Sheth – Axelar
  milap@axelar.network