ClarityAlliance

AXELAR SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA)

FEBRUARY 27TH, 2025

CONTENTS

ClarityAlliance
Security Review
Axelar

1. About Clarity Alliance

Clarity Alliance is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at clarityalliance.org.

CONTENTS

ClarityAlliance
Security Review
Axelar

2. Disclaimer

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Clarity Alliance to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Clarity Alliance's position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree
to analyze.

The assessment services provided by Clarity Alliance are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

ClarityAlliance
Security Review

Axelar

3. Introduction

A time-boxed security review of Axelar, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About Axelar

Axelar delivers secure cross-chain communication for Web3, enabling you to build Interchain dApps that grow beyond a single chain. *Secure* means Axelar is built on proof-of-stake, the battle-tested approach used by Ethereum, Polygon, Cosmos, and more. *Cross-chain communication* means you can build a complete experience for your users that lets them interact with any asset, any application, on any chain with one click.

CONTENTS

ClarityAlliance
Security Review
Axelar

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

CONTENTS

ClarityAlliance
Security Review
Axelar

6. Security Assessment Summary

- Initial analysis started at 78278c90e27ff986d21315e41c836c8125fd02c3
- Final reviewed commit 2c21ca6fc44bad6975fbefb84f64baef7fc12b3a

CONTENTS

ClarityAlliance
Security Review
Axelar

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA) engaged with - to review Axelar. In this period of time a total of **39** issues were uncovered.

Protocol Summary

Protocol Name	Axelar
Date	February 27th, 2025

Findings Count

Severity	Amount
Critical	2
High	1
Medium	3
Low	14
QA	19
Total Findings	**39**

CONTENTS

ClarityAlliance
Security Review
Axelar

Summary of Findings

ID	Title	Severity	Status
[C-01]	Token Managers Vulnerable to Draining	Critical	Resolved
[C-02]	Unauthorized Approval of Arbitrary Messages and Signer Rotation	Critical	Resolved
[H-01]	Interchain Receive Token and Execute Payload Messages Can Be Denied Execution	High	Resolved
[M-01]	Native Interchain Token Is Not SIP-10 Compliant	Medium	Resolved
[M-02]	Inflows and Outflows Are Not Accounted for When There Is No Flow Limit	Medium	Acknowledged
[M-03]	Token-ID-Claimed Event Not Emitted When Token ID Is Claimed	Medium	Resolved
[L-01]	Gas Owner Can Bypass Checks and Also Be Gas Collector	Low	Resolved
[L-02]	Proxy Calls Not Enforced for All Gas Implementation Functions	Low	Resolved
[L-03]	Silent Failures in Message Approval	Low	Resolved
[L-04]	Inadequate Contract Ownership Management	Low	Resolved
[L-05]	Interchain Operatorship Transfer Does Not Remove Flow Limiter Role	Low	Acknowledged
[L-06]	Future Gas Service Implementation Updates Will Emit Incorrect Balances	Low	Resolved
[L-07]	Missing Initialization Check in Gas Component Implementation	Low	Resolved
[L-08]	Adding and Removing Trusted Addresses Should Not Be Restricted by Pause	Low	Resolved
[L-09]	Same Contract Can Be Used for Multiple Token Deployments	Low	Resolved
[L-10]	Missing Direct Gating for Interchain Token Factory Functions	Low	Resolved
[L-11]	Potential Discrepancy in TM and NIT Deployer Identification	Low	Resolved
[L-12]	Ambiguity in Deploy Remote Interchain Token Events	Low	Resolved
[L-13]	Loss of Pending Gas Fees Upon Gas Implementation Upgrade	Low	Resolved
[L-14]	Signer Sets Do Not Expire	Low	Acknowledged

CONTENTS

ClarityAlliance
Security Review
Axelar

Summary of Findings

ID	Title	Severity	Status
[QA-01]	Typographical Errors	QA	Resolved
[QA-02]	Unspecified Flow Limit Constraint	QA	Acknowledged
[QA-03]	NIT Decimals Are Not Validated	QA	Acknowledged
[QA-04]	Verifier Upgradability Dependency	QA	Acknowledged
[QA-05]	Broken Upgradability Pattern Within Interchain Contracts	QA	Resolved
[QA-06]	Missing "Is Started" Checks in Token and Token Manager Contracts	QA	Resolved
[QA-07]	Removal of NOP-ping Internal Gas Payment	QA	Resolved
[QA-08]	Token Managers Can Self-Declare as Native Interchain Tokens	QA	Resolved
[QA-09]	Remove Debug Remnants Before Production	QA	Resolved
[QA-10]	Implement Standard Checks for All Saved Principals	QA	Resolved
[QA-11]	Revert Unimplemented Functions	QA	Resolved
[QA-12]	Overlapping Error Code Ranges	QA	Resolved
[QA-13]	Remove Dead Code	QA	Resolved
[QA-14]	Axelar Integration Chain Name Limit Bypass	QA	Resolved
[QA-15]	Add is-message-approved and is-message-executed to Gateway Proxy	QA	Resolved
[QA-16]	Enhance Code Comprehension	QA	Resolved
[QA-17]	Minor Code Optimizations	QA	Resolved
[QA-18]	ITS Implementation Should Not Be Allowed as Initial Token Minter	QA	Resolved
[QA-19]	Use Constants Where Appropriate	QA	Resolved

CONTENTS

ClarityAlliance
Security Review
Axelar

8. Findings

8.1. Critical Findings

[C-01] Token Managers Vulnerable to Draining

Description

A token manager contract is deployed for each token used in Axelar, and these managers store tokens involved in cross-chain transfers. The current implementation of the token manager has a critical flaw that allows anyone to completely drain it of tokens.

The Interchain Token Service (ITS) implementation requires, as part of its normal operations, both depositing and withdrawing tokens from the `token-manager` contract. Tokens are added through the `token-manager::give-token` function, and withdrawals are made via the `take-token` function.

Both functions are correctly restricted to be callable only by the ITS implementation, as enforced by `(asserts! (is-eq contract-caller (get-its-impl)) ERR-NOT-AUTHORIZED)`.

The issue arises from the underlying function, `transfer-token-from`, which both functions call. This function lacks proper permission checks and directly invokes the `SIP10::transfer` function on the token. This vulnerability can be exploited to drain all tokens that support authorization via `contract-caller`.

On Stacks, the SIP-10: Fungible Token Standard is somewhat ambiguous regarding the term "sender".

Older projects and tokens have interpreted "sender" to specifically mean the `tx-sender` and have implemented the transfer authorization check as: `(asserts! (is-eq tx-sender sender) (err ERR_NOT_AUTHORIZED))` (see stSTX as an example).

However, newer projects, including the sBTC token and the current Axelar interchain tokens, have chosen to also check for the contract caller:

```
(asserts! (or (is-eq tx-sender sender)
  (is-eq contract-caller sender)) ERR_NOT_OWNER)
```

In summary, any token that supports authorization via `contract-caller` can be freely drained from token managers, which primarily affects newer tokens.

Recommendation

Change the visibility of the `token-manager::transfer-token-from` function to private.

CONTENTS

ClarityAlliance
Security Review
Axelar

[C-02] Unauthorized Approval of Arbitrary Messages and Signer Rotation

Description

In the current Axelar Stacks implementation, a cross-chain message must be signed and approved by the Axelar signers before execution. Additionally, when signers are rotated, the rotation payload must be signed by the current signer set.

A critical issue arises because there is no validation to confirm that the current signer set has actually signed the provided signatures. This oversight allows anyone to sign messages and execute arbitrary commands on the chain.

The issue lies in the `gateway-impl::validate-signatures` function, where it is incorrectly assumed that calling `pub-to-signer` would map the signatures to the correct signers:

```
(signers-- (map pub-to-signer pubs signers-))
```

However, `pub-to-signer` merely returns the correct signers without validating the provided public keys or attempting to match them to any of the signers.

```
;; Helper function to iterate pubkeys along with signers and return signer
;; @param pub
;; @param signer
;; @returns {signer: (buff 33), weight: uint}
(define-private (pub-to-signer (pub (buff 33)) (signer {signer:
  (buff 33), weight: uint})) signer)
```

As a result, anyone can approve any message and rotate signers by simply signing the payload and providing it to the respective functions.
Note: The attached proof of concept (POC) demonstrates how an attacker can exploit this oversight to rotate the signers to any arbitrary set.

Recommendation

IImplement a check in `gateway-impl::validate-signatures` to ensure that all determined public keys (`pub`) are present in the existing signer set (`signers-`).

ClarityAlliance
Security Review
Axelar

8.2. High Findings

[H-01] Interchain Receive Token and Execute Payload Messages Can Be Denied Execution

Description

When an interchain message is received from the Axelar hub, it can be executed on-chain by anyone using the `execute-receive-interchain-token` function from the interchain service, once it has been approved.

This function allows for the execution of messages that either transfer tokens to a third party or execute a payload, provided it complies with the `interchain-token-executable-trait` trait. Users sending tokens from other chains and specifying a contract call on this chain will have a valid payload in the transmitted message, whereas users merely transferring tokens will have an empty data payload.

An issue arises in this design because the caller of the `execute-receive-interchain-token` message can choose whether or not to pass execution to the intended receiver. Even if the message is specifically a "receive token plus execute payload," the caller can simply ignore the execution payload and process the message with only the token transfers.

This vulnerability allows an attacker to effectively front-run all interchain execute calls and discard them. Depending on the implementation of the third-party receiver, this could lead to significant issues.

The problem occurs because, in the `interchain-token-service-impl::execute-receive-interchain-token` function, the current logic checks if either the calldata payload is empty or the destination contract is not provided, and in such cases, it completes execution successfully.

```
(if (or (is-none destination-contract) data-is-empty)
    (ok 0x)
```

Recommendation

Modify the check so that if the execution data is not empty, the destination contract must also be specified. If not, the execution should revert.

8.3. Medium Findings

[M-01] Native Interchain Token Is Not SIP-10 Compliant

Description

The implementation of the native interchain token in `native-interchain-token` does not adhere to the SIP-10 standard. Specifically, there are several issues within the `transfer` function.

The standard specifies that error codes should begin at 1 and increase incrementally, with the first four values already defined in the SIP . However, the function currently returns incorrect error codes in several instances:

| Correct Error Code | Reason | Current Incorrect Implementation Error Code | | --- | --- | --- | | u1 | `sender` does not have enough balance | ERR-INSUFFICIENT-BALANCE (err u2051) | | u2 | `sender` and `recipient` are the same principal | ERR-INVALID-PARAMS (err u2052) | | u4 | `sender` is not the same as `tx-sender` | ERR-NOT-AUTHORIZED (err u1051) |

The standard also specifies that the memo field should only be printed if it is provided:

> The implementer must ensure that the memo is emitted by adding a print statement if the `ft-transfer?` is successful and the memo is not `none`.

However, the current implementation prints an empty buffer array if there is no memo. No printing should occur in this case.

Third-party protocols may experience unexpected side effects due to these issues when integrating with any NIT token.

Recommendation

Remove the `ERR-INSUFFICIENT-BALANCE` and `ERR-INVALID-PARAMS` checks entirely, as they are already implemented in the `ft-transfer?` function.

Change the `ERR-INSUFFICIENT-BALANCE` error code to `u4` .

Modify the `print` statement so that it only triggers if the `memo` is not `none`. An example implementation from the SIP itself is: `(match memo to-print (print to-print) 0x)` .

ClarityAlliance
Security Review
Axelar

[M-02] Inflows and Outflows Are Not Accounted for When There Is No Flow Limit

Description

Both token managers and interchain tokens can have inflow and outflow limits, which are defined as:

> The maximum difference between the tokens flowing in and/or out during any given interval of time (6 hours).

Note: The following example uses interchain tokens.

If the flow limit is set to 0, it is interpreted as having no limit. This limit can also be set by any principal with the *flow-limiter* role by calling the `native-interchain-token::set-flow-limit` function.

An issue arises when no limit is set during a given epoch, as the current implementation for both inflows and outflows fails to update the `flows` map.

```
(if (is-eq limit u0)
    (ok true)
```

While the limit should not be checked if it is 0, the incoming and outgoing flows must still be accounted for.

Failing to do this results in several issues:

- External integrators that rely on the getter functions `get-flow-out-amount` and `get-flow-in-amount` to determine bridge flows will receive incorrect values.
- If, within the same epoch that the limit was removed (set to 0), flow operators reintroduce it, accounting will only resume from that point onward, leading to the following example situation:
  - An initial limit of 100,000 tokens was deemed too restrictive, so flow limiters removed it (set it to 0).
  - Unexpected market conditions cause the actual difference between inflow and outflow to reach critical levels.
  - The limit is reintroduced at 150,000.
  - At this point, users can still increase the deficit by 150,000 more, since during the no-limit period, inflows and outflows were not tracked, exacerbating the issue further.

Recommendation

In the `add-flow-out` and `add-flow-in` functions, even if the `limit` is 0, update the `flows` map with the amount changes. Implement this in both the `native-interchain-token` and `token-manager` contracts.

CONTENTS

ClarityAlliance
Security Review
Axelar

[M-03] Token-ID-Claimed Event Not Emitted When Token ID Is Claimed

Description

In the implementation of the interchain token service, the `"interchain-token-id-claimed"` event is not emitted when deploying a native interchain token using the `deploy-interchain-token` function.

This event should be emitted whenever an ID is claimed. While it is correctly emitted when a token manager is deployed, it is not emitted during the deployment of a native token.

The absence of this crucial event could lead to inconsistencies in off-chain data mechanisms.

Recommendation

In the `deploy-interchain-token` function within the `interchain-token-service-impl` contract, ensure to call the `interchain-token-service-storage::emit-interchain-token-id-claimed` function.

8.4. Low Findings

[L-01] Gas Owner Can Bypass Checks and Also Be Gas Collector

Description

In the `gas-storage` contract, when the gas collector is updated using the `set-gas-collector` function, there is a validation to ensure that the new gas collector principal is not the contract owner.

```
(asserts! (not (is-eq new-gas-collector
  (get-owner))) ERR-OWNER-CANNOT-BE-COLLECTOR)
```

However, this validation is absent when the contract owner is set through the `set-owner` function, allowing the aforementioned condition to be violated.

Recommendation

When setting the owner of the `gas-storage` contract via the `set-owner` function, ensure that the new owner is not the gas collector.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-02] Proxy Calls Not Enforced for All Gas Implementation Functions

Description

The proxy-implementation-storage pattern in use mandates that all implementation functions must be accessed via a designated proxy.

In the `gas-impl` contract, two functions, `collect-fees` and `get-balance`, permit direct calls, which violates this requirement. However, `get-balance` is a read-only function.

Recommendation

Ensure that the `collect-fees` function is accessible only through the gas service proxy contract.

ClarityAlliance
Security Review
Axelar

[L-03] Silent Failures in Message Approval

Description

In the current implementation, when a message is approved through the gateway, successful message approvals are emitted and committed in `gateway-impl::approve-message`, while any errors are ignored.

```
(map approve-message messages_)
```

If a message approval fails for any reason, external integrators cannot ascertain the cause, as `gateway::approve-messages` always returns `(ok true)`.

Recommendation

Modify the `gateway::approve-messages` function to return `(map approve-message messages_)` instead of `(ok true)`. This change will provide insight into the reasons for any failures. Additionally, in the `gateway-impl::approve-message` function, return `(ok inserted)` instead of `(ok true)` to indicate which messages were not inserted due to duplication.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-04] Inadequate Contract Ownership Management

Description

Throughout the codebase, when a contract is deployed, the deployer is identified as `(define-constant OWNER tx-sender)` or `(define-constant DEPLOYER tx-sender)`.

This principal is solely responsible for initializing the contract, even in instances where the name `OWNER` is used.

There is one exception in the `interchain-token-service-impl` contract, where the owner principal is also tasked with executing sensitive actions, such as pausing/unpausing the contract and adding or removing trusted addresses.

In this specific case, having the owner as a constant restricts flexibility and ties the contract to a single address that cannot be changed.

Recommendation

In the `interchain-token-service-impl` function, convert the `OWNER` constant into a variable to allow for changes. In all other instances, rename `OWNER` to `DEPLOYER` to better reflect its role and context.

ClarityAlliance
Security Review
Axelar

[L-05] Interchain Operatorship Transfer Does Not Remove Flow Limiter Role

Description

When a native interchain token is launched, the contract deployer must call `native-interchain-token::setup` to initialize the contract. During this process, if a valid operator principal is provided, that operator is also granted the `flow-limiter` role.

However, transferring the operator role to a different principal does not revoke this privilege, as they are typically not linked. Additionally, an operator may assign themselves the flow limiter role (add-flow-limiter) and neglect to remove it before transferring the operator privilege via `transfer-operatorship`.

This behavior may result in unauthorized addresses retaining the ability to influence the native interchain tokens.

This issue is also present in the `token-manager`, as it mirrors the functionality of the `native-interchain-token` to some extent.

Recommendation

When transferring operatorship via `transfer-operatorship`, ensure the `flow-limiter` role is also removed. Implement this change in both the `native-interchain-token` and `token-manager` contracts.

ClarityAlliance
Security Review
Axelar

[L-06] Future Gas Service Implementation Updates Will Emit Incorrect Balances

Description

When the gas service implementation is updated using the `gas-service:set-impl` function call, the balance from the previous implementation contract is retrieved and emitted:

```
(prev-balance (unwrap! (contract-call? .gas-impl get-balance) ERR-UNAUTHORIZED))
;; ...
(print {
;; ...
    balance: prev-balance
})
```

After the initial update, where this value is correctly emitted, any subsequent implementation updates will continue to display the same balance. This occurs because the previous balance is consistently retrieved from the first implementation contract, which is hardcoded as `.gas-impl`.

Recommendation

Modify the `set-impl` function to accept both the old and new implementation traits. Ensure that the old trait contract corresponds to the previous implementation and that the new trait is associated with the principal provided.

With these two traits, any necessary information can be transferred, and any setups required before losing or becoming an implementation can be executed. This solution necessitates changes to both the traits and the governance `finalize` function to allow trait passing, which may introduce a slightly high overhead.

An alternative solution is to refrain from displaying the previous balance.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-07] Missing Initialization Check in Gas Component Implementation

Description

In the gas component implementation of the codebase, there are no constraints to ensure that the setup has been called from the proxy. This oversight allows full interaction with the contracts immediately upon deployment, contrary to the intended design. The design requires the team to first call `gas-service::setup` to configure the correct `gas-collector` principal before any interaction is permitted.

Recommendation

In the `gas-impl` contract, include a check to verify that the underlying `gas-storage` component has been initialized. Ensure that `get-is-started` returns true before executing each function call.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-08] Adding and Removing Trusted Addresses Should Not Be Restricted by Pause

Description

The owner of the `interchain-token-service-impl` contract has the ability to pause or resume operations. Currently, pausing halts all operations, whether they require permission or not.

However, the actions of adding and removing trusted addresses (via the `set-trusted-address` and `remove-trusted-address` function calls) should not be affected by the pause state. These operations, along with the ability to pause the contract, are exclusively available to the contract owner.

In situations where there are issues with any trusted addresses and a pause is necessary for investigation, the contract must first be unpaused to remove an address if needed. This requirement could create a window during which other operations might be executed.

Recommendation

Eliminate the `require-not-paused` check from the `set-trusted-address` and `remove-trusted-address` functions.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-09] Same Contract Can Be Used for Multiple Token Deployments

Description

When a token manager or interchain native token is created, a token ID is generated and recorded in the storage contract.

This commitment to storage is executed through the `interchain-token-service-storage::insert-token-manager` function, which logs the newly added contract in the `token-managers` map, using the ID as the index.

The issue arises because the same deployed contract can be reused multiple times, as the ID is generated using the sender and salt:

```
keccak256( PREFIX-INTERCHAIN-TOKEN-ID | sender | salt )
```

There is no mechanism in place to check for duplicate contracts.

Whether by mistake or intentionally, the same contract can be repeatedly inserted into the contract storage system.

Recommendation

Ensure that contract addresses are unique when inserting a new token or manager in the `interchain-token-service-storage` contract.

ClarityAlliance
Security Review
Axelar

[L-10] Missing Direct Gating for Interchain Token Factory Functions

Description

The interchain token factory proxy forwards all calls to its corresponding implementation pair contract. However, the factory implementation itself lacks checks for both interchain system initialization and pause state. Most of its functions route execution through the token service proxy, which does verify component initialization (`get-is-started`) and pause state (`require-not-paused`) in its own corresponding implementation pair.

An exception to this is found in the `approve-deploy-remote-interchain-token` and `evoke-deploy-remote-interchain-token` functions within the `interchain-token-factory-impl` contract.

These functions do not verify whether the interchain component is initialized or paused.

The lack of a component initialization check has limited impact, as `approve-deploy-remote-interchain-token` would still revert with an `ERR-TOKEN-NOT-FOUND` error, since any input token provided will not exist.

However, `revoke-deploy-remote-interchain-token` can operate with non- existing and non-approved token IDs, behaving as a NOP (no-operation). This may lead to slight off-chain inconsistencies due to the emitted `"revoked-deploy-remote-interchain-token-approval"` type event.

Regarding the missing pause state check, both functions operate correctly even when the interchain component is paused, which should not occur.

Recommendation

Modify `revoke-deploy-remote-interchain-token` to check the return value of `interchain-token-service-storage::remove-approved-destination-minter` and revert if it is not true, indicating that no removal was applied. This fix will eliminate the need for an "is-started" check.

To address the pause state check, retrieve the paused status from storage and directly verify it in both `approve-deploy-remote-interchain-token` and `revoke-deploy-remote-interchain-token` .

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-11] Potential Discrepancy in TM and NIT Deployer Identification

Description

Due to limitations within the Stacks Blockchain, users must deploy interchain tokens or token managers themselves and subsequently register these contracts within the Axelar contracts as deployed.

From a semantic perspective, this registration is considered a deployment within the interchain components. However, the actual deployer of the contract is not necessarily the one recorded in the on-chain storage component.

Specifically, the individuals who call the functions to register the on-chain components are noted as the deployers. The interpretation of the deployer can vary depending on whether the APIs are accessed through the factory contracts or directly via the interchain service contract.

```
(deployer (if (is-eq caller (get-token-factory)) NULL-ADDRESS caller))
```

As a result, the deployer principal is:

1. Used to generate a unique token ID.
2. Emitted in an `interchain-token-id-claimed` event.

While generating a unique token ID may not be highly significant, discrepancies can arise if the principal who actually deployed the contract is different from the one calling the deploy token functions, leading to minor off-chain inconsistencies regarding the identity of the contract deployer.

Recommendation

For all code paths that result in the insertion of a token manager or native interchain token, ensure that the actual contract deployer is the function caller.
Specifically, in the `interchain-token-service-impl` contract, within the `deploy-token-manager` and `deploy-interchain-token` functions, if the `deployer` is not the token factory contract, verify that the decoded contract deployer `(get deployer contract-principal)` matches the `caller`.

In the `interchain-token-factory-impl` contract, implement a function equivalent to `interchain-token-service-impl::decode-contract-principal` and use it to verify the `caller` in the `register-canonical-interchain-token` and `deploy-interchain-token` functions.

These recommended changes will impose stricter constraints on token contract deployments. If this is not the intended outcome, please acknowledge this issue.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-12] Ambiguity in Deploy Remote Interchain Token Events

Description

In the `interchain-token-factory-impl` contract, there is an issue with asymmetric and duplicated events when approving or revoking the deployment of a remote interchain token.

When the `approve-deploy-remote-interchain-token` function is used to approve the deployment of a remote interchain token, an event is emitted from both the implementation contract and the storage contract, resulting in duplication.

Conversely, when revoking an approval using the `revoke-deploy-remote-interchain-token` function, only an event from the implementation contract is emitted.

To ensure consistent tracking by off-chain systems, an event should also be emitted from the storage contract when approval is revoked, similar to when it is granted.

Recommendation

Introduce a revoke event in the `interchain-token-service-storage` contract and emit it when the `revoke-deploy-remote-interchain-token` function is called.

Additionally, eliminate the duplicated event emissions from the factory implementation, retaining only those from the storage contract.

CONTENTS

ClarityAlliance
Security Review
Axelar

[L-13] Loss of Pending Gas Fees Upon Gas Implementation Upgrade

Description

When the gas component's implementation is updated, any native STX tokens remaining in the contract are lost. Although the `gas-impl` contract includes a `refund` function, it becomes inaccessible through the proxy once the official implementation is changed. Additionally, the `collect-fees` function is tied to the proxy (as discussed in a separate issue), resulting in the loss of any STX in the contract at that time.

Recommendation

In the `gas-impl::collect-fees` function, ensure the proxy call is made only if the current contract, `gas-impl` , is the active implementation. The `gas-collector` check should still be performed. This approach allows for the collection of any pending fees even after a contract update.

CONTENTS

ClarityAlliance
Security Review

Axelar

[L-14] Signer Sets Do Not Expire

Description

The signer set responsible for signing any Axelar message can be rotated under specific constraints. This rotation is intended for security purposes, allowing for continuous iteration through signers.

However, there is no actual on-chain mechanism to enforce the rotation of signers, which means any existing signer set can remain indefinitely.

Recommendation

Introduce an expiration time for each signer set.

Semantically, a rotation differs from a change, as rotation implies a cyclical event that needs to occur periodically. If the absence of enforcement is an intentional feature, this issue should be acknowledged.

8.5. QA Findings

[QA-01] Typographical Errors

Description

There are several typographical errors throughout the codebase:

- In `gas-service`:
  - At `L178`: `gas-impl-updgraded` should be corrected to `gas-impl-upgraded`.
- In `gateway-impl`:
  - At `L50`: `umambiguous` should be corrected to `unambiguous`.
  - At `L391`: `ECDS` should be corrected to `ECDSA`.
  - At `L406`: `reponse` should be corrected to `response`.
- In `gateway`:
  - At `L68`: `purose` should be corrected to `purpose`.
- In `governance`:
  - At `L200`: `governance-addres` should be corrected to `governance-address`.
- In `interchain-token-factory`:
  - At `L236`: `interchain-token-factory-impl-updgraded` should be corrected to `interchain-token-factory-impl-upgraded`.
- In `interchain-token-service`:
  - At `L433`: `interchain-token-service-impl-updgraded` should be corrected to `interchain-token-service-impl-upgraded`.
  - At `L446`: `purose` should be corrected to `purpose`.
- In `traits`:
  - At `L248`: `impls` should be corrected to `implements`.

Recommendation

Correct the identified typographical errors to enhance code consistency.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-02] Unspecified Flow Limit Constraint

Description

Interchain tokens are subject to inflow and outflow limits, which are defined as:

> The maximum difference between the tokens flowing in and/or out at any given interval of time (6h).

However, this limit is also interpreted as a maximum allowable amount for both inflow and outflow. This is because any increase in inflow or outflow cannot exceed this limit:

```
(asserts! (<= flow-amount limit) ERR-FLOW-LIMIT-EXCEEDED)
```

Due to the original intent and validation of the `flow-limit`, the maximum difference between the inflow and outflow (or vice versa) is 1 `flow-limit`. This allows for a theoretical maximum inflow or outflow amount of `2*flow-limit` while still adhering to the intended flow-limit constraint.

By restricting the variation amount to at most the limit, certain large token transfers will be blocked. Additionally, this constraint is not mentioned in the documentation.

Recommendation

Either document this behavior or remove the flow-amount to limit check if this is not intended.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-03] NIT Decimals Are Not Validated

Description

When deploying and creating a new native interchain token (NIT), the `native-interchain-token::setup` function must be called as the final step. This function includes several checks to ensure the validity of the symbol, name, and other attributes. However, it does not validate the token decimals, allowing them to be set to any arbitrary value.

Recommendation

Ensure that the `decimals_` argument in the `setup` function is validated to be greater than zero.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-04] Verifier Upgradability Dependency

Description

Whenever changes are made to the `token-manager` or `native-interchain-token` contracts, the `erify-onchain` contract, which verifies the source code of these contracts post-deployment, also requires modification.

The verifier contract is invoked by the `interchain-token-service-impl` contract, the ITS implementation. However, the contract is directly hardcoded as `.verify-onchain` rather than being passed as a trait.

This setup necessitates redeploying the implementation contract for the interchain token service each time the `token-manager` or `native-interchain-token` is updated, resulting in additional overhead and a redundant dependency.

Recommendation

Develop a verifier trait and integrate it into the execution flow until it reaches the `interchain-token-service-impl` contract, where it should be verified as the correct version. The latest, correct verifier principal can be stored either within the `interchain-token-service-impl` contract itself or in the `interchain-token-service-storage` contract.

ClarityAlliance
Security Review
Axelar

[QA-05] Broken Upgradability Pattern Within Interchain Contracts

Description

The codebase employs a three-component pattern for each of its components:

- `proxy` → serves as the main entry point for the components
- `implementation` → contains the actual business logic of each component
- `storage` → holds crucial state information. Storage contracts do not call other contracts

This pattern is intended to support contract upgradability.

Within the interchain factory and service contracts, there are two violations of this pattern.

The `interchain-token-factory-impl` contract directly calls the interchain service implementation (`interchain-token-service-impl`) through the `interchain-token-id` and `valid-token-address` functions.

The second violation occurs in the interchain token storage contract, where a function, `get-gateway` , retrieves the implementation of the gateway component. This function is never called and does not provide any value for the interchain component.

Recommendation

Incorporate the `interchain-token-id` and `interchain-token-id` functions into the interchain service proxy (and trait) and modify the `interchain-token-factory-impl` contract to call them via the service proxy.

Remove the `interchain-token-service-storage::get-gateway` function.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-06] Missing "Is Started" Checks in Token and Token Manager Contracts

Description

The `token-manager` and `native-interchain-token` contracts require initialization before they can interact with users or other protocols. Although most functions in these contracts are protected by an `is-started` check, the following functions are erroneously left unprotected:

- In `native-interchain-token`: `add-flow-limiter`, `transfer-operatorship` and `transfer-mintership`
- In `token-manager`: `add-flow-limiter` and `transfer-operatorship`

Allowing these functions to be executed before the contracts are fully initialized violates the intended design.

Recommendation

Implement an `is-started` check for all state-changing functions in both the `token-manager` and `native-interchain-token` contracts.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-07] Removal of NOP-ping Internal Gas Payment

Description

In the `interchain-token-service` contract, the `pay-native-gas-for-contract-call` function currently acts as a no-operation (NOP) when the payment amount is greater than 0. This behavior is incorrect because, in practice, gas payment is required for any operation.

The `pay-native-gas-for-contract-call` function is invoked from two locations. One instance is within the `its-hub-call-contract`, where the gas fee is already validated.

The second invocation is from the `gateway-call-contract` function. Although this function was mentioned to be removed in a different issue, if it remains, it allows calling the `pay-native-gas-for-contract-call` function with a 0 gas fee without reverting.

Recommendation

In the `pay-native-gas-for-contract-call` function within the `interchain-token-service` contract, remove the `(> amount u0)` check and directly pass the call to the `gas-service` version of the function.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-08] Token Managers Can Self-Declare as Native Interchain Tokens

Description

Once a `token-manager` contract is deployed, the deployer must invoke the `setup` unction to complete the contract's initialization.

The `setup` function enables the caller to define the token type for the contract. Currently, the system supports two types: native `TOKEN-TYPE-LOCK-UNLOCK` (for token managers) and `TOKEN-TYPE-NATIVE-INTERCHAIN-TOKEN` (for regular tokens).

Although the interchain token service component ensures that a token manager type contract should be declared as `TOKEN-TYPE-LOCK-UNLOCK`, the `token-manager::setup` function permits setting any type. If a type other than `TOKEN-TYPE-LOCK-UNLOCK` is mistakenly set, the token manager becomes inoperative.

Recommendation

Restrict the `token-manager::setup` function to only allow the `TOKEN-TYPE-LOCK-UNLOCK` type.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-09] Remove Debug Remnants Before Production

Description

The codebase contains minor debug remnants that should be removed before deployment to production.

1. In the `clarity-stacks` contract, the `debug-mode` flag is set to true, allowing sensitive operations to remain configurable. This flag should be set to `false` before production or configured to automatically set to false if the `is-mainnet` keyword returns true.
2. Developer communication remnants, such as comments labeled with (`rares:`), should be either integrated into standard function comments or removed entirely.
3. Many functions still largely reflect the Solidity Axelar implementationrather than the current Stacks version. For example, see the documentation for the `deploy-remote-canonical-interchain-token` function. Note that this issue, along with severely outdated documentation, is widespread throughout the codebase. Update all outdated comments across the codebase.
4. The `interchain-token-service-impl::is-valid-token-type` function includes a commented option within the `or` command. Remove the `or` command and the commented option.

Recommendation

Implement the specified changes.

CONTENTS

ClarityAlliance
Security Review

Axelar

[QA-10] Implement Standard Checks for All Saved Principals

Description

Within the codebase, sensitive principals are stored in the storage contracts. However, these principals are not verified to ensure they conform to the standard of the current network.

Accidentally using a testnet principal instead of a mainnet principal could make the contracts inoperative.

Recommendation

Ensure that all storage contracts saving principals verify their validity for the current network by utilizing the `-is standard` function.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-11] Revert Unimplemented Functions

Description

Within the codebase, there are occurrences of functions that are not implemented and act as NOPs (no-operations).

For example, setting governance in the `gas-service` contract using the `set-governance` function returns success, but no action is performed.

Leaving NOPs instead of implementing a revert can lead to integration confusion, such as when one contract is mistakenly used in place of another.

Recommendation

For all functions that are currently unsupported, implement a revert in their execution.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-12] Overlapping Error Code Ranges

Description

In the codebase, each contract should have a unique error code range to easily identify the contract from which the error originated. However, the current implementation of contracts uses both overlapping and interconnected ranges.

Instances of overlapping errors:

```
;; u10112
gas-impl.clar:(define-constant ERR-INVALID-AMOUNT (err u10112))
gas-storage.clar:(define-constant ERR-OWNER-CANNOT-BE-COLLECTOR (err u10112))
;; u10211
gateway.clar:(define-constant ERR-INVALID-IMPL (err u10211))
gas-service.clar:(define-constant ERR-INVALID-IMPL (err u10211))
;; u22051
interchain-token-service-impl.clar:(define-constant ERR-UNTRUSTED-CHAIN
  (err u22051))
interchain-token-service.clar:(define-constant ERR-UNTRUSTED-CHAIN (err u22051))
;; u22088
interchain-token-service.clar:(define-constant ERR-ZERO-AMOUNT (err u22088))
interchain-token-service-impl.clar:(define-constant ERR-INVALID-PARAMS
  (err u22088))
;; u4052
native-interchain-token.clar:(define-constant ERR-NOT-STARTED (err u4052))
token-manager.clar:(define-constant ERR-NOT-STARTED (err u4052))
;; u4053
native-interchain-token.clar:(define-constant ERR-UNSUPPORTED-TOKEN-TYPE
  (err u4053))
token-manager.clar:(define-constant ERR-UNSUPPORTED-TOKEN-TYPE (err u4053))
;; u5052
gateway-impl.clar:(define-constant ERR-SIGNERS-DATA (err u5052))
gateway.clar:(define-constant ERR-SIGNERS-DATA (err u5052))
;; u1051
native-interchain-token.clar:(define-constant ERR-NOT-AUTHORIZED (err u1051))
token-manager.clar:(define-constant ERR-NOT-AUTHORIZED (err u1051))
;; u2051
native-interchain-token.clar:(define-constant ERR-INSUFFICIENT-BALANCE
  (err u2051))
token-manager.clar:(define-constant ERR-FLOW-LIMIT-EXCEEDED (err u2051))
;; u2053
native-interchain-token.clar:(define-constant ERR-NOT-MANAGED-TOKEN (err u2053))
native-interchain-token.clar:(define-constant ERR-ZERO-AMOUNT (err u2053))
gateway-impl.clar:(define-constant ERR-SIGNER-WEIGHT (err u2053))
;; u21051
interchain-token-service-storage.clar:(define-constant ERR-NOT-AUTHORIZED
  (err u21051))
interchain-token-service.clar:(define-constant ERR-NOT-AUTHORIZED (err u21051))
interchain-token-service-impl.clar:(define-constant ERR-NOT-AUTHORIZED
  (err u21051))
;; u3051
native-interchain-token.clar:(define-constant ERR-FLOW-LIMIT-EXCEEDED
  (err u3051))
token-manager.clar:(define-constant ERR-NOT-MANAGED-TOKEN (err u3051))
gateway-impl.clar:(define-constant ERR-INVALID-SIGNATURE-DATA (err u3051))
;; u4051
native-interchain-token.clar:(define-constant ERR-STARTED (err u4051))
token-manager.clar:(define-constant ERR-STARTED (err u4051))
gateway-impl.clar:(define-constant ERR-INVALID-SIGNERS (err u4051))
;; u5051
native-interchain-token.clar:(define-constant ERR-ONLY-OPERATOR (err u5051))
token-manager.clar:(define-constant ERR-ONLY-OPERATOR (err u5051))
gateway-impl.clar:(define-constant ERR-INSUFFICIENT-ROTATION-DELAY (err u5051))
;; u10111
gateway-impl.clar:(define-constant ERR-UNAUTHORIZED (err u10111))
gateway-storage.clar:(define-constant ERR-UNAUTHORIZED (err u10111))
gateway.clar:(define-constant ERR-UNAUTHORIZED (err u10111))
gas-impl.clar:(define-constant ERR-UNAUTHORIZED (err u10111))
gas-service.clar:(define-constant ERR-UNAUTHORIZED (err u10111))
```

CONTENTS

ClarityAlliance
Security Review

Axelar

There are also instances where different ranges are used within a single contract, and identical ranges with different values are used across multiple contracts. Overlapping ranges between contracts can lead to confusion when debugging failed transactions.

Recommendation

Assign a distinct error range to each contract, starting from 10000 and incrementing the value for subsequent errors. The next contract in the list should start from 20000, the third from 30000, and so on.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-13] Remove Dead Code

Description

The codebase contains instances of dead code, which are sections of code that serve no purpose and can be removed.

Instances:

- In the `governance` contract, the `command-id` variable is unused in both the `execute` and `cancel` functions.
- In the `interchain-token-service` contract, the `gateway-call-contract` function is never called and is not part of any trait.

Recommendation

Remove the identified unused code.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-14] Axelar Integration Chain Name Limit Bypass

Description

The Axelar gateway integration document outlines specific constraints and recommendations for integrating chains.

The Stacks implementation has adopted limits based on the Axelar constraints.

However, Axelar explicitly requires chain names to be less than 20 characters in length:

> Chain names: The Amplifier protocol requires that chain names must be ASCII characters of length less than 20

In contrast, the Stacks implementation permits strings up to and including 20 characters in length.

Recommendation

Modify all chain representations to `(string-ascii 19)`.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-15] Add `is-message-approved` and `is-message-executed` to Gateway Proxy

Description

The gateway implementation contract includes two valuable functions, `is-message-approved` and `is-message-executed`, which are currently absent in the gateway proxy contract.

Since proxy contracts for each component are intended to serve as the sole entry points, the absence of these useful logic functions complicates usage and weakens the system architecture.

Recommendation

Incorporate `is-message-approved` and `is-message-executed` function wrappers into the `gateway` proxy contract.

ClarityAlliance
Security Review
Axelar

[QA-16] Enhance Code Comprehension

Description

The codebase contains instances where the naming conventions are slightly misleading or could be improved to enhance code comprehension.

Instances:

1. *Using* `address` *suffix for traits.*

In the `interchain-token-factory-impl::register-canonical-interchain-token` function, the term `token-address` is used to refer to a trait, not a principal (address equivalent). This is misleading because the arguments passed are not principals. The `-address` suffix should be removed in these instances.

2. *Misleading function name*

The `get-token-factory` function in the `interchain-token-service-storage` contract returns the token factory implementation, not the proxy. To better reflect its purpose, it should be renamed to `get-token-factory-impl`.

3. *Reuse existing, specific functions*

In the `token-manager` contract, the `(is-eq contract-caller (get-its-impl))` check is performed in both the `give-token` and `take-token` functions. However, there is an unused `is-its-sender` function available. This function should either be reused in these instances or removed.

In the `interchain-token-factory-impl` contract, within the `get-canonical-interchain-token-id` function, instead of calling the ITS directly, the `get-interchain-token-id-raw` function should be called with the result of `get-canonical-interchain-token-deploy-salt`.

In the same contract, within the `register-canonical-interchain-token` function, instead of calling `token-manager-address::get-token-address` and checking if `is-ok`, `token-manager-address::get-is-started` should be used directly.

Additionally, in the `approve-deploy-remote-interchain-token` function, instead of calling `interchain-token-service-storage::get-trusted-address` and checking if `is-some`, `interchain-token-service-storage::is-trusted-chain` be used.

Recommendation

Implement the suggested improvements in each case.

CONTENTS

ClarityAlliance
Security Review

Axelar

[QA-17] Minor Code Optimizations

Description

There are several opportunities for minor code optimizations throughout the codebase that can help reduce execution costs.

1. In the `governance::finalize` function, the `proxy::set-impl` and `proxy::set-governance` functions are each enclosed in an unnecessary `begin` block. Remove these redundant blocks.
2. In the `interchain-token-service-impl::execute-receive-interchain-token` function, the source-chain is retrieved from the decoded payload four times using `(get source-chain payload-decoded)`. Declare it as a variable and reuse it.
3. In the `execute-deploy-interchain-token` function of the same contract, the token-id is retrieved three times from the decoded payload using `(get token-id payload-decoded)`. Declare it as a variable and reuse it.

Recommendation

Implement the suggested code optimizations.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-18] ITS Implementation Should Not Be Allowed as Initial Token Minter

Description

For a native interchain token (NIT), the minter role permits the minting and burning of the underlying token.

During initialization, through the `native-interchain-token::setup` function, any address can be designated as the minter (if specified). However, when transferring mintership, there is a specific check to ensure that the new minter must not be the interchain token service (ITS) implementation itself.

The ITS implementation is inherently considered a minter by default, so assigning it this role is redundant.

Recommendation

In the `native-interchain-token::setup` function, ensure that the ITS is not equal to `minter_`, if specified.

CONTENTS

ClarityAlliance
Security Review
Axelar

[QA-19] Use Constants Where Appropriate

Description

To enhance code readability, it is recommended to use meaningful constants where applicable. Below are instances within the current codebases where constants can be utilized, along with suggestions:

- In `governance`;
  - At L141 and L146, the types `u1` and `u2` can be replaced with constants such as `ACTION_SET_IMPLEMENTATION` and `ACTION_SET_GOVERNANCE`.
  - At L180, the `u3` number can be replaced with a constant like `ACTION_CANCEL_TASK`.
- In `interchain-token-factory-impl` at line L143, change the `""` empty string to a constant such as `LOCAL_DEPLOYMENT`.

Recommendation

Implement the suggested changes.