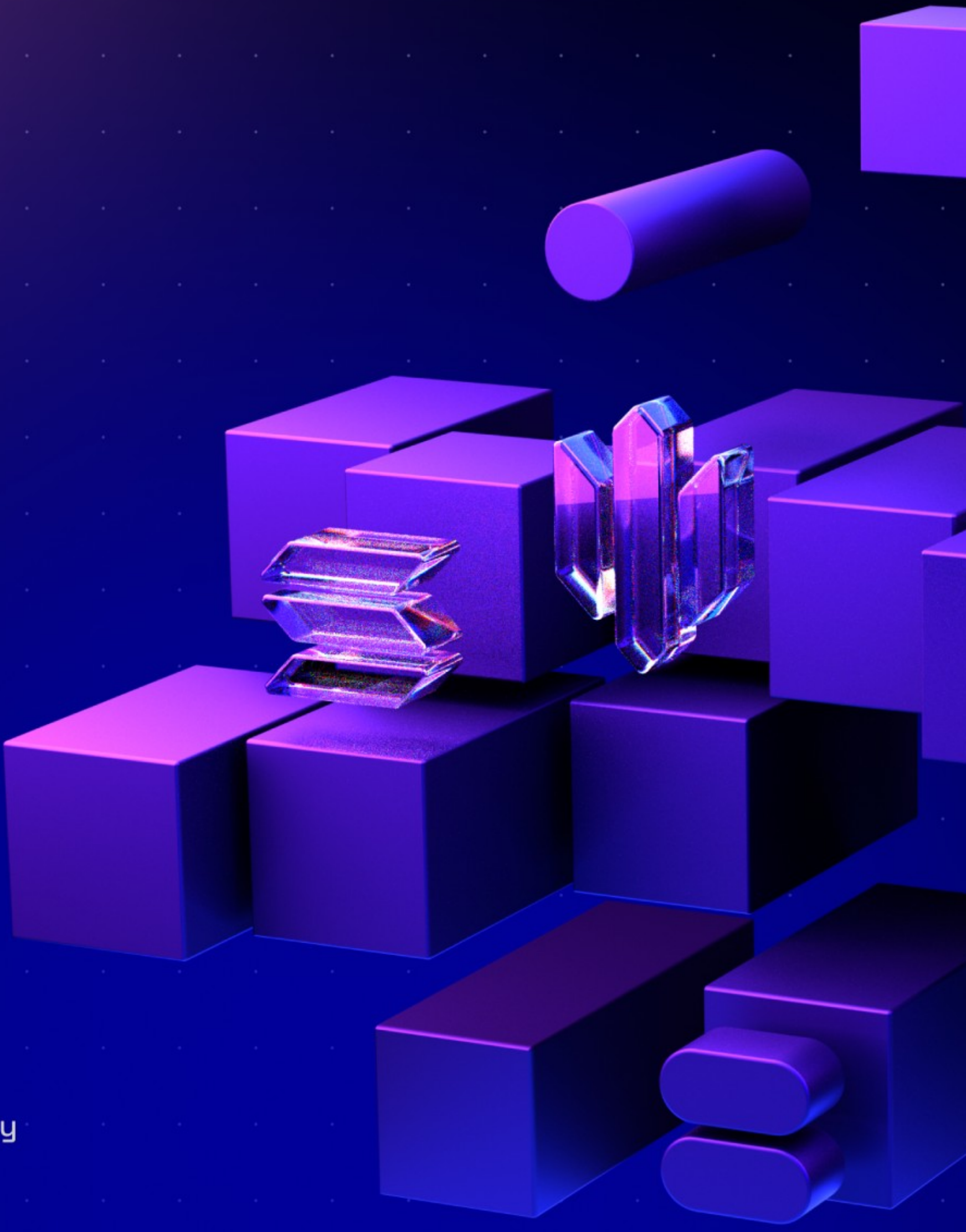
ackee
blockchain security

Axelar

Program Suite for Solana

8.8.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	25.05.2025
1.0	Final Report	26.05.2025
1.1	Fix Review	08.08.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.
Rohanske nabrezi 717/4
186 00 Prague, Czech Republic
https://ackee.xyz
hello@ackee.xyz

2.2. Audit Methodology

The Ackee Blockchain Security auditing process follows a routine series of steps:

1. **Code review**
    a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
    b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:

        missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows.

    c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.
    d. Review of best practices to improve efficiency, clarity, and maintainability.
2. **Testing and automated analysis**
    a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework Trident.
3. **Local deployment + hacking**
    a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are unique to its implementation.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Andrej Lukačovič	Lead Auditor
Matej Hyčko	Auditor
Max Kupchenko	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

The Axelar Program Suite for Solana is a suite of programs that allows General Message Passing (GMP) to the Solana blockchain, with features like Interchain Token Services (ITS), Gas payments, Governance and more.

Revision 1.0

Axelar engaged Ackee Blockchain Security to perform a security review of Axelar Program Suite for Solana with a total time donation of 43 engineering days in a period between April 7 and May 26, 2025, with Andrej Lukačovič as the lead auditor.

The audit was performed on the commit `a0dea0c`[1] and the scope was the following:

- Axelar Gas Service, excluding external dependencies;
- Axelar Gateway, excluding external dependencies;
- Axelar Interchain Token Service, excluding external dependencies;
- Axelar Multicall, excluding external dependencies;
- Axelar Governance, excluding external dependencies;
- Axelar Memo, excluding external dependencies.

We began our review by familiarizing ourselves with the codebase and documentation. The initial efforts focused on fully understanding the system, its operations, and potential vulnerability points.

In the second phase, we began direct work with the codebase. We performed in-depth analysis while creating Proof of Concept (PoC) tests for the most critical issues. During this phase, we communicated with the client to clarify system details and discuss identified issues. We paid special attention to:

- ensuring the overall protocol logic was correct and as expected;
- ensuring user funds and integrated tokens within the protocol remained safe;
- ensuring Role-based access control was properly implemented and enforced;
- ensuring instruction input validation was properly implemented throughout; and
- ensuring dedicated programs operated correctly and as expected.

In the final phase, we concluded the audit by writing the report and evaluating the severity of identified issues.

Our review resulted in 31 findings, ranging from Warning to Critical severity.

The audit revealed multiple critical severity issues.

The C1 issue allows bypass of the approval mechanism for remote deployments of tokens. This vulnerability enables an attacker to deploy legitimate tokens on the destination chain while marking themselves as a minter. Subsequently, they can mint new tokens on the destination chain and bridge them back to Solana for profit.

The C2 issue allows an attacker to create arbitrary `MessagePayload` data. Although the Axelar Interchain Token Service performs validation, this validation must be performed by the Axelar Gateway program, as this vulnerability creates critical risks for other protocols integrating with the Axelar Gateway and enables potential backdoors for third-party protocols.

The C3 issue exploits the `TokenManagerHandOverMintAuthority` instruction, allowing attackers to acquire the Minter role over any token deployed in the Axelar Interchain Token Service. The Minter role enables unlimited token minting for profit.

The C4 issue enables unauthorized token transfers from the Token Manager. Token Managers of type `LockUnlock` and `LockUnlockFee` are designed to hold tokens that are locked/unlocked during Interchain Transfer based on the transfer side. The `InterchainTransfer` instruction allows specification of the Token Manager as a source account and an attacker's wallet as the destination account, enabling Token Manager drainage and token theft.

Lastly, the C5 issue is similar to the C1 issue, but with differences in the attack vector. In this case the Solana Interchain Token Service (ITS) does not properly validate who is the creator of the deployment approval, that means an attacker can use his minter role for his worthless token, to create an approval for the legitimate token, resulting in the attacker being able to deploy the legitimate token on the destination chain and mint new tokens.

The audit also revealed multiple high and medium severity issues.

Ackee Blockchain Security recommends Axelar:

- address all reported issues;
- improve overall code quality, which is detailed in W11; and
- consider implementing all programs using the Anchor Framework to simplify validations and streamline the development process.

Ackee Blockchain Security does not recommend Axelar to immediately proceed with the deployment of the programs which were in the scope of the audit. We highly recommend addressing all issues and then performing another audit revision.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

Axelar engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

The review was performed between August 6 and August 8, 2025. The fixes were provided in separate pull requests, see the complete list of pull requests below.

Fix to the issue C1 was provided in the commit `1c245d7`[2].

Fix to the issue C2 was provided in the commit `c8b3aa9`[3].

Fix to the issue C3 was provided in the commit `fb17d14`[4].

Fix to the issue C4 was provided in the commit `4aae8d3`[5].

Fix to the issue C5 was provided in the commit `11ed62d`[6].

Fix to the issue H1 was provided in the commit `3ce983e`[7] and `e5424f8`[8].

Fix to the issue H2 was provided in the commit `267a0f4`[9].

Fix to the issue M1 was provided in the commit `509f46e`[10].

Fix to the issue M2 was provided in the commit `660f0db`[11].

Fix to the issue M3 was provided in the commit `e5424f8`[12].

Fix to the issue L1 was provided in the commit `024d096`[13].

Fix to the issue W1 was provided in the commit `25a71fc`[14].

Fix to the issue W4 was provided in the commit `6ddfb82`[15].

Fix to the issue W7 was provided in the commit `4aae8d3`[16].

From the most severe findings:

The issue C1 was partially fixed, ability to create approval by anyone allows to deploy the token by a malicious actor on destination chain.

The issue C2 was fixed, the Gateway program now validates that the committed payload hash is the same as the hash stored within the message.

The issue C3 was fixed, the Token Manager now checks that the Token Manager corresponds to the mint.

The issue C4 was fixed by removing the signer seeds from the `Transfer` instruction.

The issue C5 was fixed, the approval can be created only by the Minter over the corresponding Token Manager.

The issue H1 was fixed by making sure that the destination role account always corresponds to the resource for which the role is being transferred.

The issue H2 was fixed by adding a check to verify that the provided `operator_pda_marker_account` is initialized.

The issue M1 was fixed by updating the `create_account` instruction call to a 3-step manual process (`transfer`, `allocate`, and `assign`).

The issue M2 was fixed by making sure the program id is the same as axelar gas service program id.

The issue M3 was fixed by updating the role management process, in a way that the role hierarchy is preserved.

Additionally, one new critical issue was reported, the issue C6 allows an attacker to execute `InterchainTransfer` instruction with a Token Manager of a worthless token, while specifying the `token_id` of a legitimate token, resulting in vulnerability where the attacker would receive legitimate tokens on the destination chain, while burning/locking worthless tokens on the Solana side.

Ackee Blockchain Security recommends Axelar:

- address all reported issues;
- improve overall code quality, which is detailed in W11; and
- consider implementing all programs using the Anchor Framework to simplify validations and streamline the development process.

Ackee Blockchain Security does not recommend Axelar proceed with deployment of the audited programs. Given the severity and volume of findings from the previous revision, a comprehensive full-scope audit is required. This revision reviewed only specific fixes through individual pull requests, which provides insufficient coverage for production deployment. A complete security assessment on a frozen commit containing all fixes is necessary to ensure system integrity before production use.

[1] full commit hash: `a0dea0c0653a3bd15e7752a80c85f10cdbe2b359`

[2] full commit hash: `1c245d746b4d7bad3414778cdd3630a5ebd80fff`

[3] full commit hash: `c8b3aa9af2548921cfb1616d925999c2b434ff0f`

[4] full commit hash: `fb17d14828e47d662cee6114e14e389305ea3df5`

[5] full commit hash: `4aae8d3a4d27a793545793d451e0f056a5c1d792`

[6] full commit hash: `11ed62d37dd673c9cc7706d3a22d8c8fe82d3f1a`

[7] full commit hash: `3ce983e13b2e64e2ef4af010a06b00208b060032`

[8] full commit hash: `e5424f86ae8d8e5e9568e7461197c25f9f085fe7`

[9] full commit hash: `267a0f41be355500d8e8b16def2c11d9803292a9`

[10] full commit hash: `509f46e38cb039dbe1dbf51ab23d88d0713021e0`

[11] full commit hash: `660f0db35f91c7e8d3910ad21f5f3b103734d532`

[12] full commit hash: `e5424f86ae8d8e5e9568e7461197c25f9f085fe7`

[13] full commit hash: `024d0960453deb5e7cb33b7811ae794b680862ec`

[14] full commit hash: `25a71fcb719dee139c3982dd1e732d32c80f0ecc`

[15] full commit hash: `6ddfb826bbc9fcb09c7b3d47986577ae306a2551`

[16] full commit hash: `4aae8d3a4d27a793545793d451e0f056a5c1d792`

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- `Description`
- `Exploit scenario` (if severity is low or higher)
- `Recommendation`
- `Fix` (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
6	2	3	2	20	0	33

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
C1: Remote deployment with a specified minter can be fully bypassed with fraudulent approval	Critical	1.0	Partially fixed
C2: Axelar Solana Gateway does not ensure authenticity of Message Payload	Critical	1.0	Fixed
C3: Minter role to legitimate token can be acquired by registering fraudulent token	Critical	1.0	Fixed

Finding title	Severity	Reported	Status
C4: Token Manager can be drained by specifying attacker-controlled account as the destination	Critical	1.0	Fixed
C5: Approval of remote deployment for a particular token can be created by anyone	Critical	1.0	Fixed
H1: Possible privilege escalation for already existing role accounts	High	1.0	Fixed
H2: Operator can execute proposal without approval	High	1.0	Fixed
M1: Possible Denial of Service due to incorrect account creation	Medium	1.0	Fixed
M2: Possibility of avoiding the Gas payment due to insufficient validation of the Gas Service CPI	Medium	1.0	Fixed
M3: Possible break of roles hierarchy	Medium	1.0	Fixed
L1: Impossible to transfer authority from the update authority	Low	1.0	Partially fixed
L2: Proposal rent refunds go to the caller	Low	1.0	Reported

Finding title	Severity	Reported	Status
W1: Flow Slot is always initialized with flow amount out	Warning	1.0	Fixed
W2: Impossible to update some of the fields within the Axelar Solana Gateway Config account	Warning	1.0	Reported
W3: Incorrect Token Manager type validation	Warning	1.0	Reported
W4: Lack of validation that the Mint and Metadata accounts correspond to each other	Warning	1.0	Fixed
W5: Possibility for registration of multiple custom tokens	Warning	1.0	Reported
W6: Possibility to create any particular data account through Axelar Solana Gateway	Warning	1.0	Reported
W7: Possible undesired burn of Tokens	Warning	1.0	Fixed
W8: Transaction always reverts in case payer equals minter	Warning	1.0	Fixed
W9: Unnecessary duplication of quorum	Warning	1.0	Reported

Finding title	Severity	Reported	Status
W10: Unnecessary initialization of multiple Verifier Sets in the Axelar Solana Gateway	Warning	1.0	Reported
W11: Unusual source code architecture	Warning	1.0	Reported
W12: Time lock proposal events show unadjusted ETA values	Warning	1.0	Reported
W13: Most governance configuration fields lack update functionality	Warning	1.0	Reported
W14: Redundant rent-exemption check in withdrawal logic	Warning	1.0	Reported
W15: Instructions require unnecessary accounts	Warning	1.0	Reported
W16: Inadequate test validation	Warning	1.0	Reported
W17: Missing validation of Associated Token Account address	Warning	1.0	Reported
W18: Unrestricted refund recipient specification	Warning	1.0	Reported

Finding title	Severity	Reported	Status
W19: Cancel operator approval instruction can be called for an uninitialized approval	Warning	1.0	Reported
C6: Possibility to transfer worthless tokens on Solana side while receiving legitimate tokens on destination chain	Critical	1.1	Reported
W20: ITS Root assigned unusable OPERATOR role due to PDA signature limitations	Warning	1.1	Reported

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Andrej Lukačovič	Lead Auditor
Matej Hyčko	Auditor
Max Kupchenko	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The Axelar Program Suite for Solana is a set of programs written in Rust, dedicated to operating on the Solana blockchain. The programs are:

- Axelar Solana Gateway
- Axelar Solana Interchain Token Service
- Axelar Solana Gas Service
- Axelar Solana Governance
- Axelar Solana Multicall
- Axelar Solana Memo.

The Axelar Solana Gateway program is responsible for General Message Passing (GMP) between different blockchains. The main goal of the program is to serve as a gateway to the Solana blockchain. It can send and receive messages, which are then observed by the off-chain components and routed to the destination blockchain.

The Axelar Solana Interchain Token Service (ITS) is responsible for managing interchain tokens. The ITS builds upon the General Message Passing (GMP) functionality provided by the Axelar Solana Gateway program to manage interchain tokens. Through the ITS, tokens can be deployed directly from the source chain to the destination chain. When a token is registered in the ITS, a corresponding Token Manager is created. Users can be assigned specific roles for each Token Manager. For example, a user with a minter role for a specific Token Manager can mint new tokens of that particular token type.

The Axelar Solana Gas Service program is used to pay for interchain transactions. It acts as an intermediary between Solana programs and the Axelar network, emitting events that contain gas payment information for contract calls and managing refunds of unspent gas to Solana blockchain accounts.

The Axelar Solana Governance program enables the propagation and execution of decisions from the Axelar network to the Solana blockchain. The Axelar network schedules proposals for execution on Solana with a mandatory timelock period. After the timelock expires, any user can execute the proposal. Additionally, the program designates an operator role that, with Axelar network approval, can execute scheduled proposals before the timelock expiration.

The Axelar Solana Multicall program allows for multiple Cross-Program Invocations (CPIs) to be executed in a single transaction, based on the data provided as the instruction argument.

The Axelar Solana Memo program was developed for testing purposes, utilizing the GMP functionality of the Axelar Solana Gateway program.

Trust Model

The Axelar Program Suite for Solana implements Role-Based Access Control (RBAC) to manage access to the various program mechanisms.

In the Axelar Solana Gateway program, users must trust that:

- the Solana program `upgrade_authority` will initialize the correct `VerifierSet`; and
- the set of off-chain components will behave fairly.

In the Axelar Solana Interchain Token Service (ITS), users must trust that:

- the Solana program `upgrade_authority` will initialize and maintain the `InterchainTokenService` config as expected, without any unexpected censorship; and
- the Solana program `upgrade_authority` will appoint the correct operator to manage the Operator role over `InterchainTokenService`.

The Axelar Solana ITS contains multiple roles for Token Manager management:

- Operator
- Minter
- Flow Limiter

For these roles, users must trust that:

- the Minter will issue approvals for deployments to remote chains correctly and without unexpected censorship;
- the Minter will not unexpectedly mint new tokens into circulation with malicious intent;
- the Minter will appoint new Minters responsibly;
- the Operator will appoint new Operators responsibly;
- the Operator will appoint new Flow Limiters responsibly;
- the Operator (over `InterchainTokenService`) will set `flow_limit` for particular Token Managers fairly;
- the Flow Limiter will not unexpectedly limit the flow of tokens for particular Token Managers with malicious intent; and

- the messages from the Axelar Solana Gateway program to the Axelar Solana ITS are correctly issued and not unexpectedly censored or malformed.

In the Axelar Solana Gas Service program, users must trust that:

- funds transferred to program-controlled accounts will be used for interchain transaction payments;
- unspent gas from interchain transactions will be refunded to the recipient address specified during payment; and
- the config authority will not misappropriate funds from program-controlled accounts.

In the Axelar Solana Governance program, users must trust that the program's `upgrade_authority` will initialize the config with the correct `chain_hash` and `address_hash` values.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

C1: Remote deployment with a specified minter can be fully bypassed with fraudulent approval

Critical severity issue

Impact:	High	Likelihood:	High
Target:	programs/axelar-solana-its/src/processor/interchain_token.rs	Type:	Access control

Description

The Axelar Interchain Token Service (ITS) allows deployment of remote tokens with a specified minter on destination chains. When a minter is specified for the destination chain, the deployment must be approved by an entity that possesses the Minter role over the token on the Solana blockchain (within the Axelar ITS). However, the ITS lacks proper validation of approvals, which allows complete bypass of this approval requirement.

The following code snippet shows the `process_outbound_deploy` function from the Axelar Solana ITS, specifically the validation part of the deployment approval. The vulnerability allows an attacker to create a new deploy approval for their worthless token and use this approval for remote deployment of any particular token.

Listing 1. Excerpt from programs/axelar-solana-its/src/processor/interchain_token.rs

```
pub(crate) fn process_outbound_deploy<'a>(
    accounts: &'a [AccountInfo<'a>],
    salt: [u8; 32],
    destination_chain: String,
    maybe_destination_minter: Option<Vec<u8>>,
    gas_value: u64,
    signing_pda_bump: u8,
) -> ProgramResult {
```

```rust
    /// ...

    msg!("Instruction: OutboundDeploy");
    let destination_minter = if let Some(destination_minter) =
maybe_destination_minter {
        let minter = next_account_info(accounts_iter)?;
        let deploy_approval = next_account_info(accounts_iter)?;
        let minter_roles_account = next_account_info(accounts_iter)?;
        let token_manager_account = next_account_info(accounts_iter)?;
        outbound_message_accounts_index =
outbound_message_accounts_index.saturating_add(4);

        msg!("Instruction: OutboundDeployMinter");
        ensure_roles(
            &crate::id(),
            token_manager_account,
            minter,
            minter_roles_account,
            Roles::MINTER,
        )?;

        use_deploy_approval(minter, deploy_approval, &destination_minter)?;

        destination_minter.into()
    } else {
        Bytes::default()
    };

    /// ...
}
```

By the complete bypass of the approval mechanism, the attacker will be able to deploy any token they choose on the destination chain, with their wallet marked as the minter. This will allow them to mint the token on the destination chain and transfer it back to the Solana blockchain where it can be sold for a profit.

Exploit scenario

Bob is a legitimate user.

Alice is a malicious actor.

1. Bob deploys legitimate `TokenB` on the Solana blockchain using the `DeployInterchainToken` instruction;
2. Bob receives the Minter role over `TokenB` as part of the `DeployInterchainToken` instruction;
3. Alice deploys worthless `TokenA` on the Solana blockchain using the `DeployInterchainToken` instruction;
4. Alice receives the Minter role over `TokenA` as part of the `DeployInterchainToken` instruction;
5. Alice creates `ApprovalA` for `TokenA` by executing the `ApproveDeployRemoteInterchainToken` instruction;
6. Due to insufficient validation between the approval and the token to be deployed on the destination chain, Alice executes `DeployRemoteInterchainTokenWithMinter` with her wallet as minter, `ApprovalA` as the approval;
7. Alice successfully deploys `TokenB` on the destination chain, with her account marked as the minter;
8. Alice mints `TokenB` on the destination chain; and
9. Alice can now transfer `TokenB` from the destination chain to the Solana blockchain.

Recommendation

Ensure that the approval corresponds to the token to be deployed on the destination chain.

Partial solution 1.1

The fix provided shows only a partial fix.

Exploit scenario

Alice is a malicious actor.

Bob is a legitimate user.

1. Bob deploys legitimate `TokenB` on the Solana blockchain using the `DeployInterchainToken` instruction;
2. Bob receives the Minter role over `TokenB` as part of the `DeployInterchainToken` instruction;
3. Alice deploys worthless `TokenA` on the Solana blockchain using the `DeployInterchainToken` instruction;
4. Alice receives the Minter role over `TokenA` as part of the `DeployInterchainToken` instruction;
5. Alice creates `ApprovalB` for `TokenB` by executing the `ApproveDeployRemoteInterchainToken` instruction. Alice can create approval for `TokenB` TokenID as parameters for TokenID construction are unverified instruction inputs;
6. Alice uses the constructed approval with the `DeployRemoteInterchainTokenWithMinter` instruction, which is possible due to lack of validation that the `deployer` included in the TokenID construction is the signer;
7. Alice successfully deploys `TokenB` on the destination chain, with her account marked as the minter;
8. Alice mints `TokenB` on the destination chain; and
9. Alice can now transfer `TokenB` from the destination chain to the Solana blockchain.

Partial mitigation exists in case gas fees are paid within the `DeployRemoteInterchainTokenWithMinter` instruction. In this case, the deployer is responsible for paying the fees, which implies that the deployer has to sign.

Go back to Findings Summary

C2: Axelar Solana Gateway does not ensure authenticity of Message Payload

Critical severity issue

Impact:	High	Likelihood:	High
Target:	programs/axelar-solana-gateway/src/processor/commit_message_payload.rs	Type:	Data validation

Description

The Axelar Gateway protocol implements General Message Passing (GMP) between chains through the following process:

1. an event or message is emitted on the source chain;
2. off-chain components observe and sign the message to ensure authenticity;
3. the signed message is routed to the destination chain;
4. the Gateway program verifies signature correctness against the on-chain Verifier Set; and
5. if the quorum is reached, the message is delivered to the destination chain as valid.

The Gateway protocol extends this basic flow by:

1. using a Merkle tree of messages instead of single message, processing them in batches;
2. verifying signatures of the Merkle root;
3. validating individual messages through Merkle proof verification to ensure that the particular message is part of the Merkle tree; and

4. allowing additional message payload data to be uploaded to the Gateway program.

A critical vulnerability exists because the Gateway program does not verify that the uploaded payload's hash matches the hash stored within the message. This allows an attacker to create legitimate messages that are properly signed by off-chain entities but contain malicious payloads.

While the `validate_with_gmp_metadata` function used in the Axelar Interchain Token Service (ITS) implements this check, the Gateway program itself must enforce payload integrity as it is responsible for data correctness. This vulnerability can be exploited by third-party protocols integrating with Axelar Gateway in two ways:

1. protocols unaware of the issue may allow users to upload malicious payloads; and
2. malicious protocols could intentionally use this vulnerability as a backdoor.

Exploit scenario

Scenario 1

Alice is a third-party protocol integrating with Axelar Gateway. Bob is a malicious actor.

1. Alice is unaware of the issue and does not use `validate_with_gmp_metadata` function;
2. Bob notices the lack of validation and starts the exploit;
3. Bob sends a message from source chain to Solana chain with legitimate payload;
4. Off-chain components sign the message batch and send it to the Solana blockchain;

5. Signature verification passes as the message is properly signed;
6. Bob uploads malicious payload to the Gateway program; and
7. Bob uses the validated message with the malicious payload to execute unintended operations within Alice's protocol.

Scenario 2

Alice is a malicious protocol developer.

Alice's protocol allows deposits and withdrawals of tokens. Alice's protocol locks tokens in vault and only authorized users are able to withdraw them. Alice's protocol integrates with the Axelar Gateway protocol to deposit and withdraw tokens.

Bob is a user of Alice's protocol.

1. Alice intentionally omits the `validate_with_gmp_metadata` function check;
2. Bob uses Alice's protocol to deposit tokens effectively locking the tokens within a vault;
3. Alice uses the backdoor (i.e., not validating the payload with `validate_with_gmp_metadata` function) to substitute the legitimate payload with malicious one; and
4. Alice is able to execute withdrawals from Bob's corresponding vault.

Recommendation

Ensure that the Gateway program validates that the committed payload hash is the same as the hash stored within the message.

Fix 1.1

The issue is fixed by comparing the uploaded payload during commit to the expected hash in the Incoming Message.

Go back to Findings Summary

C3: Minter role to legitimate token can be acquired by registering fraudulent token

Critical severity issue

Impact:	High	Likelihood:	High
Target:	programs/axelar-solana-its/src/processor/token_manager.rs	Type:	Access control

Description

The Axelar Interchain Token Service (ITS) allows registration of existing tokens on the Solana blockchain through the `RegisterCustomToken` instruction. After registration, the `TokenManagerHandOverMintAuthority` instruction transfers the original mint authority to the Token Manager.

A critical vulnerability exists in the `handover_mint_authority` function's validation logic. While the function validates the Token Manager's Program Derived Address (PDA), it only checks against the provided `token_id` parameter without verifying the relationship between the Token Manager and the mint address. The relevant code shows:

Listing 2. Excerpt from programs/axelar-solana-its/src/processor/token_manager.rs

```rust
pub(crate) fn handover_mint_authority(
    accounts: &[AccountInfo<'_>],
    token_id: [u8; 32],
) -> ProgramResult {
    // ...
    assert_valid_token_manager_pda(
        token_manager,
        its_root.key,
        &token_id,
        token_manager_config.bump,
    )?;
    // ...

}
```

After validation, the instruction transfers mint authority and sets up roles:

Listing 3. Excerpt from programs/axelar-solana-its/src/processor/token_manager.rs

```rust
COption::Some(authority) if authority == *payer.key => {
    let authority_transfer_ix = spl_token_2022::instruction::set_authority(
        token_program.key,
        mint.key,
        Some(token_manager.key),
        AuthorityType::MintTokens,
        payer.key,
        &[],
    )?;
    invoke(&authority_transfer_ix, &[mint.clone(), payer.clone()])?;
    setup_roles(
        payer,
        token_manager,
        payer.key,
        minter_roles,
        system_account,
        Roles::MINTER,
    )?;
    Ok(())
}
```

This insufficient validation allows an attacker to: 1. register their own token; 2. obtain the `token_id` of a target token; and 3. use the target token's Token Manager while transferring their own token's mint authority, gaining unauthorized minting privileges.

Exploit scenario

Bob is a malicious actor.

TokenTarget is a target token.

1. Bob creates a new token (TokenB) on the Solana blockchain;
2. Bob calls the `RegisterCustomToken` instruction to register TokenB;
3. Bob uses the Solana explorer to obtain the `token_id` of his target token (TokenTarget), `token_id` is stored within the TokenManager account;
4. Bob calls the `TokenManagerHandOverMintAuthority` instruction, providing the Token Manager account corresponding to TokenTarget but with the TokenB mint address;
5. Due to insufficient account validation, the instruction transfers the mint authority of TokenB to the Token Manager, but most importantly, the instruction will also grant Bob the Minter role over the Token Manager corresponding to TokenTarget; and
6. Bob can now mint unlimited tokens and sell them for profit.

Recommendation

Ensure that the Token Manager used in the `TokenManagerHandOverMintAuthority` instruction corresponds to the token for which the minter role is being transferred.

Fix 1.1

The issue is fixed by adding a check that the Token Manager corresponds to the mint.

Go back to Findings Summary

C4: Token Manager can be drained by specifying attacker-controlled account as the destination

Critical severity issue

Impact:	High	Likelihood:	High
Target:	programs/axelar-solana-its/src/processor/interchain_transfer.rs	Type:	Access control

Description

The Axelar Interchain Token Service supports Interchain Transfers with different Token Manager types. For Token Manager types `LockUnlock` and `LockUnlockFee`, tokens are locked on the source chain rather than burned, with the Token Manager holding these tokens in its Token Account.

A critical vulnerability exists due to two implementation flaws:

1. Insufficient validation of destination and source Token Accounts during instruction execution, allowing attackers to specify their own address as the destination while using the Token Manager's Token Account as the source; and
2. Incorrect specification of signer seeds for the `Transfer` instruction, as shown in the following code:

Listing 4. Excerpt from programs/axelar-solana-its/src/processor/interchain_transfer.rs

```rust
let signers_seeds: &[&[u8]] = if accounts.authority.key ==
accounts.token_manager_pda.key {
    token_manager_pda_seeds
} else {
    &[]
};
```

```rust
let transferred = match token_manager.ty {
    NativeInterchainToken | MintBurn | MintBurnFrom => {
        /// ...
    }
    LockUnlock => {
        let decimals = get_mint_decimals(accounts.token_mint)?;
        let transfer_info =
            create_take_token_transfer_info(accounts, amount, decimals, None,
signers_seeds);
        transfer_to(&transfer_info)?;
        amount
    }
    LockUnlockFee => {
        let (fee, decimals) = get_fee_and_decimals(accounts.token_mint,
amount)?;
        let transfer_info = create_take_token_transfer_info(
            accounts,
            amount,
            decimals,
            Some(fee),
            signers_seeds,
        );
        transfer_with_fee_to(&transfer_info)?;
        amount
            .checked_sub(fee)
            .ok_or(ProgramError::ArithmeticOverflow)?
    }
};
```

Although the program expects tokens to be transferred to the Token Manager for locking, it incorrectly provides Token Manager authorization seeds. This allows the transfer instruction to move funds from the Token Manager’s Associated Token Account to any attacker-specified account.

Exploit scenario

Alice is a malicious actor.

TokenTarget is a target token. TokenTarget has Token Manager of type `LockUnlock`.

1. Alice discovers, using a Solana explorer, a Token Manager of type `LockUnlock` holding locked tokens;
2. Alice crafts an Interchain Transfer instruction with destination address set to her own account and source address set to the Token Manager's Token Account;
3. Alice executes the instruction, successfully transferring the locked tokens to her account; and
4. Alice can repeat this process to drain all tokens from the Token Manager.

Recommendation

Ensure that the Token Manager cannot be the source of the Interchain Transfer instruction. Ensure there are no signer seeds used in unnecessary places.

Fix 1.1

The issue is fixed by removing the signer seeds from the `Transfer` instruction.

Go back to Findings Summary

C5: Approval of remote deployment for a particular token can be created by anyone

Critical severity issue

Impact:	High	Likelihood:	High
Target:	programs/axelar-solana-its/src/processor/interchain_token.rs	Type:	Access control

Description

The Axelar Interchain Token Service (ITS) allows deploying tokens on other chains using remote deployment instructions. When an account should be marked as a minter on the destination chain, the current minter (i.e., the account possessing the minter role on Solana) needs to approve the remote deployment.

The issue arises from the fact that the approval can be created by anyone. The `ApproveDeployRemoteInterchainToken` instruction lacks validation that the Token Manager and its corresponding minter are actually connected to the token which is going to be deployed. This allows for an attack vector similar to the issue described in C1, where an attacker can deploy a token on another chain, mint new tokens, and bridge them back to Solana.

Exploit scenario

Bob is a legitimate user.

Alice is a malicious actor.

1. Bob deploys legitimate `TokenB` on the Solana blockchain using the `DeployInterchainToken` instruction;

2. Bob receives the Minter role over `TokenB` as part of the `DeployInterchainToken` instruction;
3. Alice deploys worthless `TokenA` on the Solana blockchain using the `DeployInterchainToken` instruction;
4. Alice executes `ApproveDeployRemoteInterchainToken` where she creates `ApprovalB` to approve the deployment of `TokenB` on the destination chain, she uses her Minter role over `TokenA` to create the approval;
5. Alice executes `DeployRemoteInterchainTokenWithMinter` with her wallet as minter, `ApprovalB` as the approval;
6. Alice successfully deploys `TokenB` on the destination chain, with her account marked as the minter;
7. Alice mints `TokenB` on the destination chain; and
8. Alice can now transfer `TokenB` from the destination chain to the Solana blockchain.

Recommendation

Ensure that the remote deployment approval can be created only by the Minter over the corresponding Token Manager and the approval corresponds to the token to be deployed on the destination chain.

Fix 1.1

The issue is fixed by allowing only the Minter over the corresponding Token Manager to create the approval.

Go back to Findings Summary

H1: Possible privilege escalation for already existing role accounts

High severity issue

Impact:	High	Likelihood:	Medium
Target:	helpers/role-management/src/processor.rs	Type:	Access control

Description

The Axelar Interchain Token Service (ITS) implements Role-Based Access Control (RBAC) with the following roles:

- `OPERATOR`;
- `MINTER`; and
- `FLOW_LIMITER`.

Token Manager

The `OPERATOR` over the Token Manager is able to appoint new `FLOW_LIMITER` over the Token Manager using the `TokenManagerAddFlowLimiter` instruction.

The `OPERATOR` over the Token Manager is able to transfer `OPERATOR` role from one account to another using the `TokenManagerTransferOperatorship` instruction.

The `MINTER` over the Token Manager is able to mint tokens to any account using the `InterchainTokenMint` instruction.

The `MINTER` over the Token Manager is able to transfer `MINTER` role from one account to another using the `InterchainTokenTransferMintership` instruction.

The `MINTER` over the Token Manager is able to create approvals for remote deployment if a minter is specified for the destination chain using the `ApproveDeployRemoteInterchainToken` instruction.

The `FLOW_LIMITER` over the Token Manager is able to set `flow_limit` in the Token Manager using the `TokenManagerSetFlowLimit` instruction.

Interchain Token Service Config

The `OPERATOR` over the Interchain Token Service Config account is able to transfer other `OPERATOR` roles to other accounts using the `OperatorTransferOperatorship` instruction.

The `OPERATOR` over the Interchain Token Service Config account is able to add `flow_limit` in the Token Manager using the `SetFlowLimit` instruction.

A high severity vulnerability exists in the `role_management::processor::transfer` function, which handles role transfers between accounts. The function lacks proper validation of the destination role account being modified. If a user already had any role for the corresponding Token Manager, the transfer will be able to escalate roles even to the highest privilege level. This insufficient validation creates two significant security issues:

1. Users can escalate their existing roles to higher privilege levels; and
2. Users can restore previously revoked roles, as the `UserRoles` account remains active even after role removal.

Exploit scenario

Alice is a regular user.

Bob is a malicious actor.

1. Alice deploys `TokenA` as a legitimate token;

2. Alice grants `FLOW_LIMITER` role to Bob over the `TokenA` Token Manager;
3. Bob deploys `TokenB` as a worthless token;
4. Bob exploits the `role_management::processor::transfer` vulnerability by executing the `TokenManagerTransferOperatorship` instruction, using the destination role account that corresponds to his `FLOW_LIMITER` role on `TokenA`;
5. The instruction adds the `OPERATOR` role to Bob's `UserRoles` account;
6. Bob can now appoint new `MINTER` over the `TokenA` Token Manager; and
7. Bob can now mint `TokenA` tokens to any account.

Recommendation

Ensure that the destination role account always corresponds to the resource for which the role is being transferred.

Fix 1.1

The issue was fixed by making sure that the destination role account always corresponds to the resource for which the role is being transferred.

Go back to Findings Summary

H2: Operator can execute proposal without approval

High severity issue

Impact:	High	Likelihood:	Medium
Target:	execute_operator_proposal.rs, operator.rs	Type:	Data validation

Description

The `ExecuteOperatorProposal` instruction allows operators to execute scheduled proposals before their estimated time of arrival (ETA) and without proper approval. While the instruction verifies that the provided `operator_pda_marker_account` address matches the authorized operator's address, it fails to validate whether the `operator_pda_marker_account` is initialized.

Listing 5. Excerpt from operator

```
let calculated_pda = Pubkey::create_program_address(
    &[
        seed_prefixes::OPERATOR_MANAGED_PROPOSAL,
        proposal_hash,
        &[proposal.managed_bump()],
    ],
    &crate::ID,
)?;
if calculated_pda != *proposal_managed_pda.key {
    msg!("Derived operator managed proposal PDA does not match provided 
one");
    return Err(ProgramError::InvalidArgument);
}
```

Exploit scenario

Alice, a malicious governance operator, attempts to execute a proposal immediately after its scheduling:

1. A proposal is scheduled with a time lock;
2. Alice calls the `execute_operator_proposal` instruction with the correct `operator_pda_marker_account` address; and
3. The proposal executes successfully before its ETA, bypassing the time lock protection.

Recommendation

Add validation to ensure the `operator_pda_marker_account` is initialized before allowing proposal execution. This validation confirms that the operator has proper authorization to execute proposals before their ETA.

Fix 1.1

The issue was fixed by adding a check to verify that the provided `operator_pda_marker_account` is initialized.

Go back to Findings Summary

M1: Possible Denial of Service due to incorrect account creation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	helpers/program-utils/src/lib.rs	Type:	Denial of service

Description

The Axelar programs implement insufficient account initialization logic by relying solely on the `system_program::create_account` instruction. This instruction fails when the destination account has a non-zero balance, creating a vulnerability in the protocol.

An attacker can prevent the initialization of any desired account by transferring a minimum of 1 lamport to the target account address before its creation. This attack vector enables a Denial of Service attack against the protocol's account creation mechanism.

Exploit scenario

Alice, a malicious actor, can prevent the creation of critical protocol accounts:

1. Alice monitors the Axelar Gateway program on the Ethereum chain;
2. Alice obtains the bridging message, including the `command_id`;
3. Alice derives the `IncomingMessage` Program Derived Address (PDA) that the Axelar Gateway program on Solana would derive;
4. Alice transfers 1 lamport to the `IncomingMessage` PDA; and
5. The Axelar Gateway program on Solana fails to execute the `ApproveMessage`

instruction because the `IncomingMessage` PDA has a non-zero balance.

Recommendation

Use the `create_account` instruction only when the balance of an account is 0.

Use manual `transfer`, `assign`, and `allocate` instructions in case the balance is not 0.

Fix 1.1

The issue is fixed by updating the `create_account` instruction call to a 3-step manual process (`transfer`, `allocate`, and `assign`).

Go back to Findings Summary

M2: Possibility of avoiding the Gas payment due to insufficient validation of the Gas Service CPI

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	programs/axelar-solana-its/src/processor/gmp.rs	Type:	Data validation

Description

For the Interchain Transfers, Axelar Interchain Token Service (ITS) supports Cross-Program Invocation (CPI) into the Axelar Gas Service program. The Gas Service program allows payment for the gas of interchain transactions on the source chain.

In the following code, we can see the basic idea of the gas payment mechanism for the interchain communication. In case the `gas_value` is greater than 0, the `pay_gas` function is called.

Listing 6. Excerpt from programs/axelar-solana-its/src/processor/gmp.rs

```rust
pub(crate) fn process_outbound<'a>(
    payer: &'a AccountInfo<'a>,
    accounts: &GmpAccounts<'a>,
    payload: &GMPPayload,
    destination_chain: String,
    gas_value: u64,
    signing_pda_bump: u8,
    payload_hash: Option<[u8; 32]>,
    wrapped: bool,
) -> ProgramResult {
    // ...
    if gas_value > 0 {
        pay_gas(
            payer,
            accounts.gas_service,
            accounts.gas_service_config_account,
            accounts.system_program,
            payload_hash,
            its_root_config.its_hub_address,
            gas_value,
        )?;
    }
    // ...
}
```

Below, the Cross-Program Invocation (CPI) into the Axelar Gas Service program can be seen.

Listing 7. Excerpt from programs/axelar-solana-its/src/processor/gmp.rs

```rust
fn pay_gas<'a>(
    payer: &'a AccountInfo<'a>,
    gas_service: &'a AccountInfo<'a>,
    gas_service_config: &'a AccountInfo<'a>,
    system_program: &'a AccountInfo<'a>,
    payload_hash: [u8; 32],
    its_hub_address: String,
    gas_value: u64,
) -> ProgramResult {
    let gas_payment_ix =
axelar_solana_gas_service::instructions::pay_native_for_contract_call_instruc
tion(
            gas_service.key,
            payer.key,
            gas_service_config.key,
            crate::ITS_HUB_CHAIN_NAME.to_owned(),
            its_hub_address,
            payload_hash,
            *payer.key,
            vec![],
            gas_value,
        )?;

    invoke(
        &gas_payment_ix,
        &[
            payer.clone(),
            gas_service_config.clone(),
            system_program.clone(),
        ],
    )
}
```

The main issue is that Axelar ITS lacks validation of the Gas Service program that is going to be invoked. This vulnerability allows an attacker to create their own malicious Gas Service program which will mimic the original Gas Service behavior without actually transferring the lamports (i.e., gas). Additionally, based on the off-chain relayer implementation, this can cause a critical severity issue as, based on the emitted data from the malicious Gas Service, the relayer will think that the gas was paid and will forward the transaction to the destination chain.

The relayer code is out of the scope of this audit, but from the brief validation of the source code, we suspect that the relayer is not validating the Gas Service program address that was executed, resulting in the vulnerability mentioned above.

The issue was discussed with the client.

> *Ackee Team: Is the relayer listening to all logs that mention the specified gas_service_config_pda or is there any filtering based on the Gas Service address ?*
>
> *Axelar Team A1: Hello, there is no filtering on the gas service address. The gas service and the relayer are currently in a 1-1 relationship. There's only one gas service deployed, and each relayer, points to a specific gas service config pda (specified on the relayer config).*
>
> *Axelar Team A2: A little correction from our side. We are actually checking the gas service id when listening to logs.*
>
> — Axelar - Ackee Team

NOTE

As the discussion above shows the communication between the Ackee and the Axelar Team, we initially evaluated the issue with high likelihood as we received clarification (A1) that the logs are not sufficiently verified. The answer was corrected (A2) and source code where the validation happens was provided (the validation is part of the relayer code, which is out of the scope of this audit). Thus, we decided to reduce the likelihood to low.

Exploit scenario

Bob is a malicious actor.

1. Bob creates a similar copy of the Axelar Gas Service program, mimicking the original behavior, however without the lamports transfer;
2. Bob uses the malicious Gas Service program during execution of some of the Interchain instructions, for example `InterchainTransfer` instruction;
3. The malicious program will emit into the logs that the gas was paid, however without actual lamports transfer;
4. The `InterchainTransfer` is finalized on the destination chain without actual lamports transfer; and
5. Bob effectively executed the `InterchainTransfer` instruction without paying the gas.

Recommendation

Properly validate Cross-Program Invocation (CPI) into the Axelar Gas Service program.

Fix 1.1

The issue is fixed by making sure the program id is the same as axelar gas service program id.

[Go back to Findings Summary]

M3: Possible break of roles hierarchy

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	-	Type:	Code quality

Description

As mentioned in the H1, the Axelar Interchain Token Service (ITS) implements Role-Based Access Control (RBAC) for the Token Manager. The design specifies a strict role hierarchy where the `OPERATOR` role has the highest privileges and can assign other roles, while the `MINTER` role should be limited to token minting operations.

However, this hierarchy is not properly enforced. A vulnerability in the Axelar ITS allows the `MINTER` role to manipulate the `OPERATOR` role due to insufficient validation in the `InterchainTokenTransferMintership` instruction.

NOTE The roles management implementation uses an unconventional approach. Role transfers between users require multiple steps, making the process unnecessarily complex.

Exploit scenario

Bob is a malicious actor.

1. Bob is a `MINTER` over the `TokenA` Token Manager;
2. Bob executes the `InterchainTokenTransferMintership` instruction, with the `OPERATOR` role being transferred and his role accounts as the destination; and
3. Bob freely becomes an `OPERATOR` over the `TokenA` Token Manager.

Recommendation

Review the process of role management and simplify it. Instead of transferring the role from one user to another, create a new set of instructions dedicated for each role, where roles are simply added or removed. Remove role types from the instruction inputs as these are most of the time compared to single values and always make sure that the hierarchy is preserved.

Fix 1.1

The issue was fixed by updating the role management process, in a way that the role hierarchy is preserved.

Go back to Findings Summary

L1: Impossible to transfer authority from the update authority

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	programs/axelar-solana-its/src/processor/mod.rs	Type:	Configuration

Description

The Axelar Interchain Token Service (ITS) provides several instructions for updating the on-chain state of the ITS Config Account:

- `SetPauseStatus`;
- `SetTrustedChain`; and
- `RemoveTrustedChain`.

While the protocol implements Role-Based Access Control (RBAC) for role management, the above instructions to update the on-chain state can only be executed by the `update_authority`. This design creates two significant issues:

1. if the Axelar ITS program is marked as final (i.e., the upgrade authority is set to None), the `update_authority` will not be present, making it impossible to update the on-chain state; and
2. when a new `update_authority` is appointed, access to the above instructions is automatically transferred to the new `update_authority`, which might not be the intended behavior.

Exploit scenario

Alice is a program deployer, the Axelar Team.

1. Alice deploys the Axelar ITS program;
2. After some time Alice decides to mark the programs as final, removing the `update_authority` from the program; and
3. Alice is not able to update the on-chain state, as the `update_authority` is not present.

Recommendation

Implement the authority transfer mechanism to another wallet. Allow only the `Initialize` instruction to be executed by the `update_authority`.

Partial solution 1.1

The issue was partially fixed by allowing the `OPERATOR` role over the ITS Root to set and remove trusted chains.

The `SetPauseStatus` instruction remains accessible only by the `update_authority`.

Go back to Findings Summary

L2: Proposal rent refunds go to the caller

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	execute_proposal.rs	Type:	N/A

Description

When a proposal is executed after its ETA, the proposal account is closed and its rent lamports are refunded to the account of the user calling the instruction. This means that anyone can execute proposals after their ETA and receive the rent refunds, rather than having these lamports returned to a protocol-controlled account.

Exploit scenario

Alice observes that proposal rent refunds go to the caller.

1. Alice creates a bot to monitor for proposals that have not yet reached their ETA;
2. the bot waits for each proposal to become eligible and executes them immediately after their ETA; and
3. Alice accumulates rent refunds from multiple proposal accounts over time.

Recommendation

Consider directing rent refunds to a protocol-controlled account.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W1: Flow Slot is always initialized with flow amount out

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-its/src/processor/interchain_transfer.rs	Type:	Logic error

Description

The Axelar Interchain Token Service (ITS) tracks token flows during corresponding epochs defined within the ITS. The Token Manager contains a variable called `flow_limit`, which helps to balance between token amounts coming in and out of the Solana chain. If the `flow_limit` is set to 0, there is no limit on the token flow.

The following code shows how the flow is tracked:

Listing 8. Excerpt from programs/axelar-solana-its/src/processor/interchain_transfer.rs

```rust
fn track_token_flow(
    accounts: &FlowTrackingAccounts<'_>,
    flow_limit: u64,
    amount: u64,
    direction: FlowDirection,
) -> ProgramResult {
    if flow_limit == 0 {
        return Ok(());
    }

    let current_flow_epoch = flow_limit::current_flow_epoch()?;
    if let Ok(mut flow_slot) = FlowSlot::load(accounts.flow_slot_pda) {
        // ...

        flow_slot.add_flow(flow_limit, amount, direction)?;
        flow_slot.store(
            accounts.payer,
            accounts.flow_slot_pda,
            accounts.system_account,
        )?;
    } else {
        // ...

        let flow_slot = FlowSlot::new(flow_limit, 0, amount,
flow_slot_pda_bump)?;
        flow_slot.init(
            // ...

        )?;
    }

    Ok(())
}
```

The issue arises in the flow slot account initialization logic. When the flow slot account cannot be loaded, a new account is initialized. However, regardless of the transfer direction, the new account is always initialized with `flow_in` set to 0 and `flow_out` set to the transfer amount. This initialization pattern is incorrect as it does not consider the actual direction of the transfer.

Recommendation

Ensure that the flow slot account is initialized with the correct flow amounts based on the transfer direction.

Fix 1.1

The issue is fixed by initializing the FlowSlot with the correct amount based on the direction of the transfer.

Go back to Findings Summary

W2: Impossible to update some of the fields within the Axelar Solana Gateway Config account

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-gateway/src/processor.rs	Type:	Configuration

Description

The Axelar Solana Gateway program creates a GatewayConfig account to store the gateway configuration. However, some of the fields are set during the config initialization and cannot be updated later. More precisely, the following fields are immutable:

- `previous_verifier_retention`;
- `minimum_rotation_delay`;
- `operator`; and
- `domain_separator`.

If the fields cannot be updated, they should be turned into constants stored within the program.

Recommendation

Decide if the fields should be immutable and if so, turn them into constants. If not, implement the logic to allow their update.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W3: Incorrect Token Manager type validation

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-its/src/processor/token_manager.rs	Type:	Data validation

Description

The Axelar Interchain Token Service (ITS) implements the `validate_token_manager_type` function to validate token manager types. The current implementation contains several validation issues:

Listing 9. Excerpt from programs/axelar-solana-its/src/processor/token_manager.rs

```rust
pub(crate) fn validate_token_manager_type(
    ty: token_manager::Type,
    token_mint: &AccountInfo<'_>,
    token_manager_pda: &AccountInfo<'_>,
) -> ProgramResult {
    let mint_data = token_mint.try_borrow_data()?;
    let mint = StateWithExtensions::<Mint>::unpack(&mint_data)?;

    match (mint.base.mint_authority, ty) {
        (COption::None, token_manager::Type::NativeInterchainToken) => {
            msg!("Mint authority is required for native InterchainToken");
            Err(ProgramError::InvalidInstructionData)
        }
        (COption::Some(key), token_manager::Type::NativeInterchainToken)
            if &key != token_manager_pda.key =>
        {
            msg!("TokenManager is not the mint authority, which is required for this token manager type");
            Err(ProgramError::InvalidInstructionData)
        }
        (_, token_manager::Type::LockUnlockFee)
            if !mint
                .get_extension_types()?
                .contains(&ExtensionType::TransferFeeConfig) =>
        {
            msg!("The mint is not compatible with the LockUnlockFee
TokenManager type, please make sure the mint has the TransferFeeConfig
extension initialized");
            Err(ProgramError::InvalidAccountData)
        }
        _ => Ok(()),
    }
}
```

The function has the following issues:

1. The mint authority requirement is not enforced for all relevant token manager types:
   - Required for `NativeInterchainToken`;
   - Missing validation for `MintBurnFrom`; and
   - Missing validation for `MintBurn`.
2. The unlimited supply validation branch only covers the `NativeInterchainToken` type, omitting necessary checks for:
   - `MintBurnFrom` type; and
   - `MintBurn` type.
3. During `RegisterCustomToken` execution, users can specify the token manager type through instruction input. When registering a token as `NativeInterchainToken`, the token manager will typically not have mint authority, leading to potential validation failures.

Recommendation

Make sure the validation is exhaustive and covers all the token manager types. Update the validation for cases where the token manager is not the mint authority, during the `RegisterCustomToken` instruction.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W4: Lack of validation that the Mint and Metadata accounts correspond to each other

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-its/src/processor/interchain_token.rs	Type:	Data validation

Description

The Axelar Interchain Token Service (ITS) program allows for deployment of new tokens on the destination chain. The instructions dedicated for the remote deployment expect Mint account and the corresponding Metadata account on the instruction input.

The issue arises due to lack of validation between the Mint account and its corresponding Metadata account, which results in a situation where any Metadata account and Mint account can be used.

The following code snippet shows the basic idea of how the Interchain Message for remote deployment is constructed:

Listing 10. Excerpt from programs/axelar-solana-its/src/processor/interchain_token.rs

```rust
pub(crate) fn process_outbound_deploy<'a>(
    accounts: &'a [AccountInfo<'a>],
    salt: [u8; 32],
    destination_chain: String,
    maybe_destination_minter: Option<Vec<u8>>,
    gas_value: u64,
    signing_pda_bump: u8,
) -> ProgramResult {
    let mint = next_account_info(accounts_iter)?;
    // ...

    let token_metadata = Metadata::from_bytes(&metadata.try_borrow_data()?)?;
    let mint_data = Mint::unpack(&mint.try_borrow_data()?)?;

    let message = GMPPayload::DeployInterchainToken(DeployInterchainToken {
        selector: DeployInterchainToken::MESSAGE_TYPE_ID
            .try_into()
            .map_err(|_err| ProgramError::ArithmeticOverflow)?,
        token_id: token_id.into(),
        name: token_metadata.name.trim_end_matches('\0').to_owned(),
        symbol: token_metadata.symbol.trim_end_matches('\0').to_owned(),
        decimals: mint_data.decimals,
        minter: destination_minter,
    });

    // ...
}
```

The lack of validation between the Mint and Metadata accounts can cause:

- incorrectly set decimal points for the remote token; and
- incorrect token name and symbol for the remote token.

Recommendation

Ensure that the Mint and Metadata accounts correspond to each other.

Fix 1.1

The issue was fixed by adding a validation that the Mint and Metadata accounts correspond to each other.

Go back to Findings Summary

W5: Possibility for registration of multiple custom tokens

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-its/src/processor/link_token.rs	Type:	Logic error

Description

The Axelar Interchain Token Service (ITS) allows for registration of already existing tokens. Two categories of tokens are allowed: `Canonical` and `Custom`.

Important to note is that the Token Manager address during token registration is derived from the `token_id` of the token.

The `Canonical` tokens have `token_id` derived from the token's address. This means it is possible to create only one Token Manager for the corresponding token.

However, the `Custom` tokens have `token_id` derived from the `payer` account address and the `salt` provided on the instruction input. This means it is possible to create multiple Token Managers for the same token.

We mark this as a warning, as even if there will be multiple token managers for the token, the attacker will not have mint authority over the registered token, nor will the attacker be able to call the `TokenManagerHandOverMintAuthority` instruction as it is not expected that the attacker is the current mint authority.

Recommendation

Ensure that tokens can be registered only once.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W6: Possibility to create any particular data account through Axelar Solana Gateway

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-gateway/src/processor/initialize_message_payload.rs	Type:	Logic error

Description

The Axelar Gateway program's message verification process relies on Merkle trees to verify cross-chain messages. After message verification, the program allows payload creation through two main instructions:

1. `InitializeMessagePayload`: creates a new `MessagePayload` account; and
2. `WriteMessagePayload`: writes an unverified bytearray payload to this account.

A security concern arises because Axelar programs do not implement Account Discriminators to identify different account types. As a result, an attacker can leverage these two instructions to initialize arbitrary data accounts owned by the Gateway program, potentially creating any desired data structure.

Recommendation

For all data accounts, implement Account Discriminator to enforce uniqueness of all account types.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W7: Possible undesired burn of Tokens

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-its/src/processor/interchain_transfer.rs	Type:	Data validation

Description

As described in the C4, the Axelar Interchain Token Service (ITS) supports multiple Token Manager types. For Token Managers of type `NativeInterchainToken`, `MintBurn`, or `MintBurnFrom`, tokens are burned on the Solana chain during the Interchain Transfer.

The following code snippet demonstrates the token burning implementation:

Listing 11. Excerpt from programs/axelar-solana-its/src/processor/interchain_transfer.rs

```
fn handle_take_token_transfer(
    accounts: &TakeTokenAccounts<'_>,
    token_manager: &TokenManager,
    amount: u64,
) -> Result<u64, ProgramError> {

    // ...

    let signers_seeds: &[&[u8]] = if accounts.authority.key ==
accounts.token_manager_pda.key {
        token_manager_pda_seeds
    } else {
        &[]
    };

    let transferred = match token_manager.ty {
        NativeInterchainToken | MintBurn | MintBurnFrom => {
            burn(
                accounts.authority,
                accounts.token_program,
                accounts.token_mint,
                accounts.source_account,
                amount,
                signers_seeds,
            )?;
            amount
        }
        // ...
    };

    Ok(transferred)
}
```

A vulnerability exists when `signers_seeds` is used for the `burn` instruction. If the Token Manager holds any tokens, these tokens can be burned by specifying the Token Manager's Associated Token Account as the source account.

This issue is classified as a `Warning` because Token Managers of these types are not designed to hold tokens. Furthermore, there is no incentive for malicious users to send tokens to the Token Manager's Associated Token Account, as doing so would result in burning their own tokens.

Recommendation

Ensure that it is not possible to burn tokens from the Token Manager's Associated Token Account. Remove unnecessary signer seeds, as the amount is burned from the user's account, meaning that the Token Manager is not required to be a signer.

Fix 1.1

The issue was fixed by removing the `signers_seeds` parameter from the `burn` instruction.

[Go back to Findings Summary]

W8: Transaction always reverts in case payer equals minter

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-its/src/processor/interchain_token.rs	Type:	Logic error

Description

The Axelar Interchain Token Service (ITS) enables token deployment on destination chains through the `DeployRemoteInterchainTokenWithMinter` instruction. During execution, the `process_outbound_deploy` function verifies minter approval for remote deployment when a destination chain minter address is specified.

The issue arises due to incorrect placement of the `close_pda` account. The `close_pda` function manually transfers lamports from the approval account to the rent recipient. This manual transfer interferes with the Solana runtime, as the runtime does not recognize the manual lamport balance adjustment, leading to transaction reversion when the same account is used as input in subsequent CPIs.

Recommendation

Ensure that the `close_pda` is executed at the end of the instruction. Additionally, verify that moving the `close_pda` function to the end of the instruction does not break the existing logic.

Fix 1.1

The issue was fixed by moving the `close_pda` function to the end of the instruction.

Go back to Findings Summary

W9: Unnecessary duplication of quorum

Impact:	Warning	Likelihood:	N/A
Target:	crates/axelar-solana-encoding/src/types/verifier_set.rs	Type:	Code quality

Description

The Axelar Gateway program requires a quorum (minimum number of signatures) for message verification and approval. The current implementation passes this quorum value as a parameter in the `SigningVerifierSetInfo` structure to the `VerifySignature` instruction.

This approach creates unnecessary data duplication that could be avoided by storing the quorum value on-chain. The `VerifierSetTracker` account could store this value, following standard blockchain development practices. This modification would eliminate redundant data transmission in each verification request and ensure consistent quorum requirements across all `SigningVerifierSetInfo` instances.

Recommendation

Store the quorum value on-chain in the `VerifierSetTracker` account instead of passing it as an instruction parameter.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W10: Unnecessary initialization of multiple Verifier Sets in the Axelar Solana Gateway

Impact:	Warning	Likelihood:	N/A
Target:	programs/axelar-solana-gateway/src/processor/initialize_config.rs	Type:	Code quality

Description

During the Axelar Gateway `InitializeConfig` instruction, multiple `VerifierSets` accounts can be initialized. This behavior is unnecessary and potentially confusing, as initializing a single `VerifierSet` account during the `InitializeConfig` instruction is sufficient.

Recommendation

Remove the loop that enables initialization of multiple `VerifierSets` accounts, as only one account is needed.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W11: Unusual source code architecture

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Code quality

Description

The Axelar programs exhibit poor code quality and lack several best practices commonly used in Solana programs written without the Anchor framework:

1. The programs lack account discriminators, which is crucial for scenarios where data accounts can have the same length. As described in W6, the program allows creation of any particular data account with almost any length.
2. Account validation is not consistently implemented. Instruction inputs lack proper validation, resulting in undesired behavior. For example, insufficient validation during the role transfer instruction as described in M3.
3. Validation logic is fragmented across multiple functions and locations. As described in C2, the validation is not implemented in the appropriate place.
4. Program Derived Address (PDA) seeds follow an unusual pattern. For example, the `IncommingMessage` and corresponding `MessagePayload` accounts are not directly connected, but both use `command_id` as a seed.
5. The codebase contains several instances of redundant implementations. The validation of the signer flag is unnecessarily duplicated, and some instruction input accounts are unnecessarily duplicated while others remain unused.

The following code snippet shows the unnecessary validation of the signer flag. The `verification_session_account` account is Program Derived Address

(PDA) and is not expected to be a signer:

Listing 12. Excerpt from programs/axelar-solana-gateway/src/processor/initialize_payload_verification_session.rs

```rust
pub fn process_initialize_payload_verification_session(
    program_id: &Pubkey,
    accounts: &[AccountInfo<'_>],
    merkle_root: [u8; 32],
) -> ProgramResult {
    // ...
    // Check verification session account requirements
    if verification_session_account.is_signer {
        solana_program::msg!("Error: verification session account is not a signer");
        return Err(ProgramError::InvalidAccountData);
    }
    // ...

}
```

The following code snippet demonstrates unused accounts in the instruction input. The instruction loads 3 accounts while 5 are skipped (`OUTBOUND_MESSAGE_ACCOUNTS_INDEX`):

Listing 13. Excerpt from programs/axelar-solana-gateway/src/processor/initialize_payload_verification_session.rs

```rust
pub(crate) fn process_outbound_deploy<'a>(
    accounts: &'a [AccountInfo<'a>],
    salt: [u8; 32],
    destination_chain: String,
    maybe_destination_minter: Option<Vec<u8>>,
    gas_value: u64,
    signing_pda_bump: u8,
) -> ProgramResult {
    const OUTBOUND_MESSAGE_ACCOUNTS_INDEX: usize = 5;
    let accounts_iter = &mut accounts.iter();
    let payer = next_account_info(accounts_iter)?;
    let mint = next_account_info(accounts_iter)?;
    let metadata = next_account_info(accounts_iter)?;
    let mut outbound_message_accounts_index =
    OUTBOUND_MESSAGE_ACCOUNTS_INDEX;

    // ...
}
```

These architectural and implementation issues collectively reduce confidence in the source code quality.

Recommendation

Review and refactor the source code to address the identified issues. Consolidate all instruction input validations, including both account and parameter validations, into a single location. Remove unused accounts from instruction inputs and replace variable input parameters with constants where they should have fixed values.

Additionally, simplify the Program Derived Address (PDA) seeds by removing unnecessary seed overheads. Clean up the codebase by removing unused or redundant code, including duplicate validations and unused instruction accounts.

Update 1.1

The issue remains unresolved.

Go back to Findings Summary

W12: Time lock proposal events show unadjusted ETA values

Impact:	Warning	Likelihood:	N/A
Target:	schedule_time_lock_proposal.rs	Type:	Logging

Description

The `ScheduleTimeLockProposal` instruction correctly enforces a minimum ETA for proposals by adjusting any ETA below the minimum threshold. However, when this adjustment occurs, the instruction emits an event containing the original, unadjusted ETA value instead of the actual ETA used in the proposal.

Listing 14. Excerpt from schedule_time_lock_proposal

```
// Send event
let event = GovernanceEvent::ProposalScheduled {
    hash: ctx.proposal_hash,
    target_address: ctx.target.to_bytes(),
    call_data: ctx.cmd_payload.call_data.into(),
    native_value: ctx.cmd_payload.native_value.to_le_bytes(),
    eta: ctx.cmd_payload.eta.to_le_bytes(),
};
```

Recommendation

Modify the event emission to use the final, adjusted ETA value that was applied to the proposal instead of the original value from the instruction parameters.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W13: Most governance configuration fields lack update functionality

Impact:	Warning	Likelihood:	N/A
Target:		Type:	Configuration

Description

Once the governance configuration is initialized, only the `operator` field can be modified. While some fields like `bump` should remain constant since they are tied to the singleton config account, other fields such as `minimum_proposal_delay_eta` may need to be adjusted in the future.

Recommendation

Add functionality to update configuration fields that may require modification in the future, such as `minimum_proposal_delay_eta`.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W14: Redundant rent-exemption check in withdrawal logic

Impact:	Warning	Likelihood:	N/A
Target:	withdraw_tokens.rs	Type:	Unused code

Description

The program performs a manual verification to ensure the config account maintains rent-exempt status after withdrawal. This check is redundant since the Solana runtime automatically verifies rent exemption at the end of each transaction and fails any transaction that would make an account non-rent-exempt.

Listing 15. Excerpt from withdraw_tokens

```rust
// Ensure we do not go below the rent-exempt balance
let rent = Rent::get()?;
let resultant_amount_after_operation = config_pda
    .lamports()
    .checked_sub(amount)
    .expect("to not overflow when calculating
resultant_amount_after_operation");

if resultant_amount_after_operation <
rent.minimum_balance(config_pda.data_len()) {
    msg!("Not enough lamports to keep the account alive");
    return Err(ProgramError::InsufficientFunds);
}
```

Recommendation

Remove the manual rent-exemption check as it consumes unnecessary compute units and duplicates the runtime's built-in verification.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W15: Instructions require unnecessary accounts

Impact:	Warning	Likelihood:	N/A
Target:	transfer_operatorship.rs	Type:	Code quality

Description

Multiple instructions in the governance program require accounts that are never used in their implementation. For instance, in the `TransferOperatorship` instruction, both the `system_account` and `payer` accounts are specified as required but remain unused throughout the instruction's execution.

Listing 16. Excerpt from transfer_operatorship

```
let accounts_iter = &mut accounts.iter();
let _system_account = next_account_info(accounts_iter)?;
let _payer = next_account_info(accounts_iter)?;
let operator_account = next_account_info(accounts_iter)?;
let config_pda = next_account_info(accounts_iter)?;
```

Recommendation

Remove the unused accounts from the instruction definitions to improve code clarity.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W16: Inadequate test validation

Impact:	Warning	Likelihood:	N/A
Target:	tests/module/execute_operator_proposal.rs	Type:	Code quality

Description

The test suite contains tests that pass despite not properly validating critical functionality. For example, a test intended to verify that an operator cannot execute an unapproved proposal passes regardless of whether the proposal was approved by the GMP or not.

This suggests that other tests may also have insufficient validation checks, potentially masking issues in the codebase.

Recommendation

Perform a comprehensive review of the test suite with focus on validation assertions. Ensure each test properly validates the specific scenario it is designed to test.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W17: Missing validation of Associated Token Account address

Impact:	Warning	Likelihood:	N/A
Target:	spl.rs	Type:	Data validation

Description

The `ensure_valid_config_pda_ata` function performs basic token account validation by checking the token program owner, mint, and account owner, but does not verify that the provided account is actually an Associated Token Account (ATA). While the function name and usage context suggest that an ATA should be used, any token account that satisfies these basic checks can be used instead of the proper ATA.

Listing 17. Excerpt from spl

```rust
fn ensure_valid_config_pda_ata(
    config_pda_ata: &AccountInfo<'_>,
    token_program: &AccountInfo<'_>,
    mint: &AccountInfo<'_>,
    config_pda: &AccountInfo<'_>,
) -> ProgramResult {
    if config_pda_ata.owner != token_program.key {
        return Err(ProgramError::IncorrectProgramId);
    }
    let ata_data =
spl_token_2022::state::Account::unpack_from_slice(&config_pda_ata.try_borrow_
data()?)?;
    if ata_data.mint != *mint.key || ata_data.owner != *config_pda.key {
        return Err(ProgramError::InvalidAccountData);
    };
    Ok(())
}
```

Exploit scenario

1. Alice, a malicious user, creates a regular SPL token account with the same mint as the expected token.
2. Alice transfers ownership of this token account to the config PDA.
3. Alice passes this regular token account to functions expecting an ATA.
4. The validation succeeds since `ensure_valid_config_pda_ata` only checks the program owner, mint, and account owner, allowing the use of a non-ATA token account where an ATA was intended.

Recommendation

Add validation in the `ensure_valid_config_pda_ata` function to verify that the provided token account address matches the canonical ATA derivation for the given mint and owner.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W18: Unrestricted refund recipient specification

Impact:	Warning	Likelihood:	N/A
Target:	spl.rs, native.rs	Type:	Trust model

Description

The program can send refunds to any address specified in the transaction input. There are no restrictions on who can receive the refund, which could lead to funds being sent to unintended recipients.

Exploit scenario

In this scenario, Alice is a malicious actor attempting to exploit the unrestricted refund functionality.

1. Alice deploys a program that emits events mimicking the gas service program's contract call payment functionality;
2. Alice's program emits an event that matches the structure of a legitimate gas service payment, including a valid config PDA account and Alice's address as the refund recipient;
3. Off-chain components process this event as legitimate since it contains a valid config PDA account and matches the expected event structure;
4. Alice requests a refund from the program without having made an actual payment; and
5. The program processes the refund and transfers funds to Alice's address since refund recipients are not validated against on-chain data.

NOTE

During discussions with the client, it was clarified that this scenario is mitigated by the relayer off-chain component, which was outside the audit scope. The component verifies that events originate exclusively from the gas service program.

> Nevertheless, we still recommend implementing the on-chain validation for an additional layer of security.

Recommendation

Create a PDA at the time of contract call payment to store the refund recipient address on-chain. Use this stored address when processing refunds instead of accepting an arbitrary address from the transaction input.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

W19: Cancel operator approval instruction can be called for an uninitialized approval

Impact:	Warning	Likelihood:	N/A
Target:	cancel_operator_approval.rs	Type:	Data validation

Description

The `CancelOperatorApproval` instruction succeeds even when attempting to cancel a non-existent approval. This is caused by two issues:

1. The instruction has the same validation issue as H2, where it does not verify whether the provided `operator_proposal_pda` is initialized; and
2. The `close_pda` function succeeds even when the provided PDA account is uninitialized.

Recommendation

Add a validation check within the `close_pda` function to verify that the provided PDA account is initialized before attempting to close it.

Update 1.1

The issue was reported and remains unaddressed.

Go back to Findings Summary

Report Revision 1.1

Revision Team

Revision team is the same as in Report Revision 1.0.

Overview

Since there were no comprehensive changes in this revision, the complete overview is listed in the Executive Summary section Revision 1.1.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

C6: Possibility to transfer worthless tokens on Solana side while receiving legitimate tokens on destination chain

Critical severity issue

Impact:	High	Likelihood:	High
Target:	-	Type:	Data validation

Description

The `InterchainTransfer` instruction of the Axelar Interchain Token Service allows transferring tokens from one chain to another. The instruction accepts `token_id` as a parameter, which is later emitted as a parameter for the destination chain indicating which tokens should be received.

The vulnerability occurs due to lack of validation of the `token_id` parameter. Even though the instruction provides some necessary checks, it does not check if the `token_id` is valid. This can result in a situation where an attacker burns or locks worthless tokens on the Solana side while receiving legitimate tokens on the destination chain.

The following code snippet shows where the vulnerability occurs. The `token_id` is passed as a parameter to the instruction. Later in the instruction, the Token Manager address is verified; however, this address does not take into consideration the `token_id` parameter, but the `token_id` stored within the Token Manager. Lastly, after the amount is burned/locked, the event is constructed containing the unverified `token_id` parameter.

Listing 18. Excerpt from programs/axelar-solana-its/src/processor/interchain_transfer.rs

```rust
pub(crate) fn process_outbound_transfer<'a>(
    accounts: &'a [AccountInfo<'a>],
    token_id: [u8; 32],
    // ...
) -> ProgramResult {

    msg!("Instruction: OutboundTransfer");
    let token_manager =
TokenManager::load(take_token_accounts.token_manager_pda)?;
    assert_valid_token_manager_pda(
        take_token_accounts.token_manager_pda,
        take_token_accounts.its_root_pda.key,
        &token_manager.token_id,
        token_manager.bump,
    )?;

    let amount_minus_fees = take_token(&take_token_accounts, &token_manager,
amount)?;
    amount = amount_minus_fees;

    // ...


    let transfer_event = event::InterchainTransfer {
        token_id, // token_id is not verified
        source_address: *take_token_accounts.token_manager_ata.key,
        destination_chain,
        destination_address,
        amount,
        data_hash: if let Some(data) = &data {
            if data.is_empty() {
                [0; 32]
            } else {
                solana_program::keccak::hash(data.as_ref()).0
            }
        } else {
            [0; 32]
        },
    };

    // ...

    let payload = GMPPayload::InterchainTransfer(InterchainTransfer {
        selector: InterchainTransfer::MESSAGE_TYPE_ID
            .try_into()
            .map_err(|_err| ProgramError::ArithmeticOverflow)?,
        token_id: token_id.into(), // token_id is not verified
        source_address: take_token_accounts.token_mint.key.to_bytes().into(),
        destination_address: transfer_event.destination_address.into(),
        amount: alloy_primitives::U256::from(amount),
        data: data.unwrap_or_default().into(),
    });
}
```

Exploit scenario

Alice is a malicious actor.

Bob is a legitimate user.

1. Bob deploys legitimate `TokenB` on the Solana blockchain;
2. Alice deploys worthless `TokenA` on the Solana blockchain;
3. Alice, by using any available Token Explorer, looks for the `token_id` of `TokenB`;
4. Alice calls the `InterchainTransfer` instruction with the `token_id` of `TokenB` as a parameter, but with the Token Manager of `TokenA`;
5. Alice burns/locks (depends on Token Manager type) tokens of `TokenA` on the Solana side;
6. Alice receives `TokenB` on the destination chain due to the `token_id` parameter corresponding to `TokenB`; and
7. Alice effectively transfers worthless tokens on the Solana side while receiving legitimate tokens on the destination chain.

Recommendation

Ensure that the tokens being locked or burned on the Solana side are the same as the tokens being received on the destination chain.

Go back to Findings Summary

W20: ITS Root assigned unusable OPERATOR role due to PDA signature limitations

Impact:	Warning	Likelihood:	N/A
Target:	-	Type:	Logic error

Description

The `DeployInterchainToken` instruction of the Axelar Interchain Token Service (ITS) allows to deploy new Token and its Token Manager on the Solana blockchain. The instruction assigns the `OPERATOR` role over the Token Manager to the ITS Root. However, ITS Root is a Program Derived Address (PDA), which means the address can only sign Cross-Program Invocation (CPI) instructions. The `OPERATOR` role for the ITS Root is therefore not usable, as each instruction over Token Manager operating with this role requires the role owner to sign the transaction.

Recommendation

Ensure the ITS Root can use the `OPERATOR` role over the Token Manager, or remove the role from the ITS Root.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Axelar: Program Suite for Solana, 8.8.2025.

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz