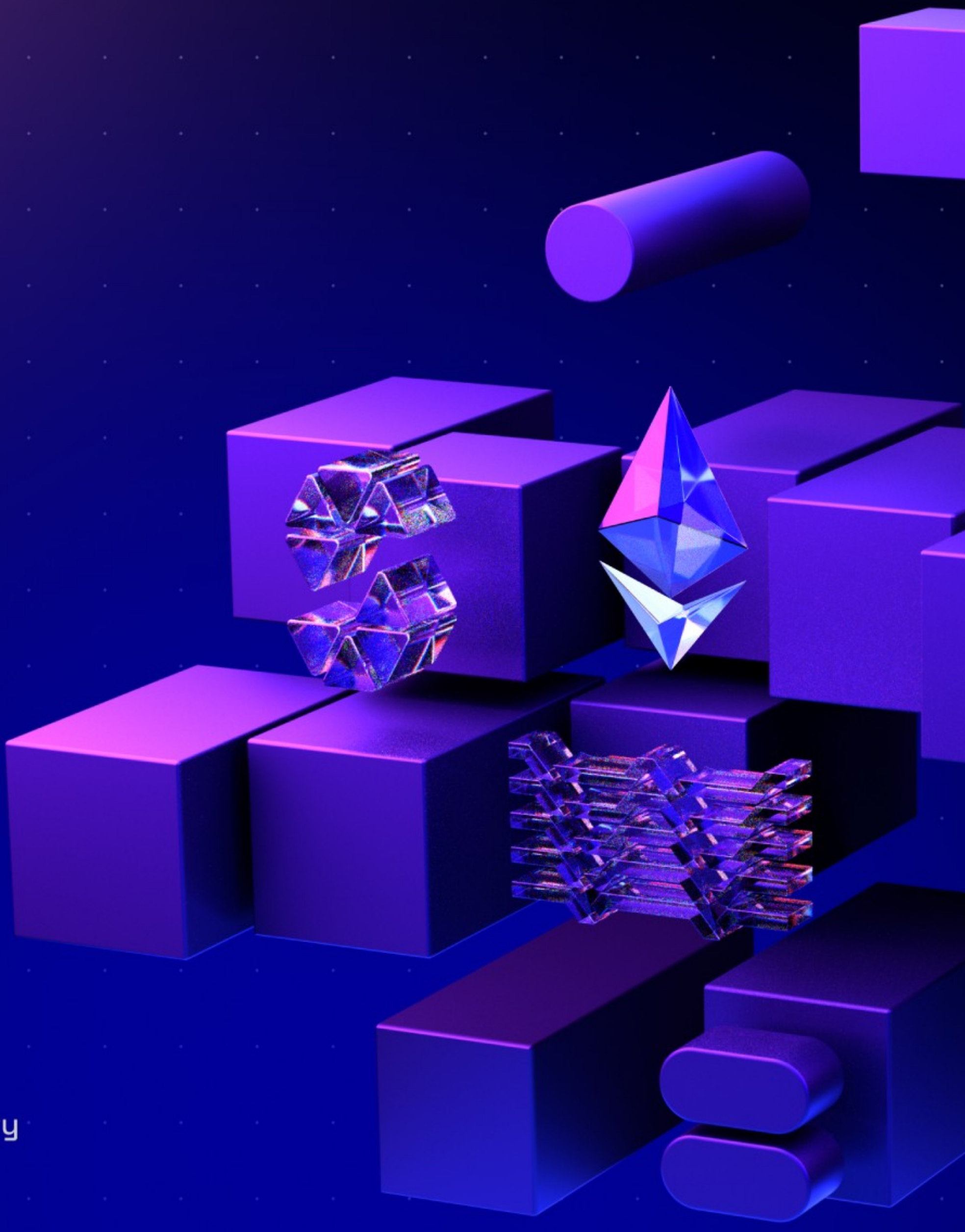
ackee
blockchain security

Axelar

Interchain Token Service

16.1.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	29.11.2024
1.0	Final Report	04.12.2024
1.1	Fix review	16.01.2025

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

1. **Verification of technical specification**

   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**

   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**

   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**

   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**

   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

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
Lukáš Rajnoha	Lead Auditor
Jan Převrátil	Auditor
Michal Převrátil	Auditor
Martin Veselý	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

Revision 1.0

Axelar engaged Ackee Blockchain Security to perform a security review of the Axelar Interchain Token Service protocol with a total time donation of 18 engineering days in a period between October 29 and November 29, 2024, with Lukáš Rajnoha as the lead auditor.

The audit was performed on the commit `7e6916`[1] and the scope was the following:

- `InterchainTokenFactory.sol`
- `InterchainTokenService.sol`
- `TokenHandler.sol`
- `utils/TokenManagerDeployer.sol`
- `utils/InterchainTokenDeployer.sol`
- `utils/Create3AddressFixed.sol`
- `utils/GatewayCaller.sol`
- `utils/Create3Fixed.sol`
- `token-manager/TokenManager.sol`
- `types/InterchainTokenServiceTypes.sol`.

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we involved the Wake testing framework. During the review, we paid special attention to:

- newly introduced ITS Hub interoperability;

- ensuring interchain token transfers work as intended;
- looking for issues in custom token managers;
- ensuring correct mintership privileges in interchain tokens;
- ensuring access controls are not too relaxed;
- proper upgradeability patterns in ITS contracts;
- looking for common issues such as data validation.

Our review resulted in 17 findings, ranging from Info to Medium severity. The most severe one, M1, concerns missing recovery mechanisms for locked funds when interchain transfers fail while utilizing the ITS Hub. L3 and L2 highlight potential issues with non-standard and fee-on-transfer `ERC-20` tokens.

Ackee Blockchain Security recommends Axelar:

- reconsider proper recovery mechanisms for locked funds when utilizing the ITS Hub;
- assess support for non-standard and fee-on-transfer `ERC-20` tokens;
- address all other reported issues.

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

Revision 1.1

Axelar engaged Ackee Blockchain to perform a fix review on the given commit: `a56fd79` [2].

From 17 findings, 6 issues were fixed and 11 acknowledged. The status of all reported issues has been updated and can be seen in the findings table.

See Report Revision 1.1 for the revision overview.

[1] full commit hash: `7e6916d0d5bfe0c2c1bc915cc1ac5b21f82d1b36`

[2] full commit hash: a56fd79f0e507c0803d59062ac5b7f83954f0722

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
0	0	1	3	5	8	17

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
M1: Funds Cannot Be Retrieved from Failed Interchain Transactions	Medium	1.0	Acknowledged
L1: Empty Token Deployment in `InterchainTokenFactory.deployInterchainToken` Function	Low	1.0	Fixed
L2: Optional `ERC-20` Functions Required	Low	1.0	Acknowledged
L3: Tokens with Callbacks Can Break Accounting	Low	1.0	Acknowledged

Finding title	Severity	Reported	Status
W1: `GatewayCaller` Contract Ambiguously Reverts on Insufficient Funds	Warning	1.0	Acknowledged
W2: Missing Zero Address Check for Token in `TokenManagerProxy`	Warning	1.0	Fixed
W3: Factory Does Not Check If Canonical Token Exists When Registering It	Warning	1.0	Fixed
W4: `TokenManager` Function Selector Clashes	Warning	1.0	Partially fixed
W5: Multiple Contracts Receive Ether Without Withdrawal Capability	Warning	1.0	Acknowledged
I1: Implementation Details of `RolesBase` Exposed in `TokenManager`	Info	1.0	Acknowledged
I2: Duplicate Message Type Definitions	Info	1.0	Acknowledged
I3: Inconsistent Type Usage for `TokenManagerType`	Info	1.0	Acknowledged
I4: Excessive Inheritance in `InterchainToken`	Info	1.0	Acknowledged
I5: Typos or Missing Documentation	Info	1.0	Acknowledged
I6: Function `TokenManager.tokenAddress` Can Be Marked as Pure	Info	1.0	Acknowledged

Finding title	Severity	Reported	Status
I7: `InterchainTokenService.execute` Function Placement Inconsistency	Info	1.0	Fixed
I8: Unused Code	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Lukáš Rajnoha	Lead Auditor
Jan Převrátil	Auditor
Michal Převrátil	Auditor
Martin Veselý	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The Interchain Token Service is a modular cross-chain token bridging protocol built on Axelar Network. It enables seamless token transfers between blockchains through token managers that handle canonical ERC-20s and `InterchainTokens` with native cross-chain capabilities. The system supports both lock/unlock and mint/burn mechanics while maintaining consistent token properties across chains. The `InterchainTokenFactory` may provide additional guarantees like fixed supply and removed deployer privileges.

Trust Model

The Interchain Token Service inherits the security of Axelar GMP and the new ITS Hub. Users of the protocol have to trust the Axelar infrastructure to perform interchain relayings. In the context of this project, users have to trust the Axelar team to deploy and setup the `InterchainTokenService` contract and its proxies correctly. Users have to trust canonical `ERC-20` tokens on source chains.

Fuzzing

The fuzzing was performed with a total donation of 3 engineering days with Michal Převrátil as the fuzz test implementer. During the fuzzing process, no additional issues were discovered.

A manually-guided differential stateful fuzz test was developed during the review to test the correctness and robustness of the system. The differential fuzz test keeps its own Python state according to the system's specification. Assertions are used to verify the Python state against the on-chain state in contracts. The list of all implemented execution flows and invariants is available in Appendix B. These tests covered:

- interchain token deployments and canonical token registrations via the `InterchainTokenFactory` contract;
- interchain transfers in the `InterchainTokenService` contract, simulating both GMP and ITS Hub cross-chain relaying.

Findings

The following section presents the list of findings discovered in this revision.

M1: Funds Cannot Be Retrieved from Failed Interchain Transactions

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	various	Type:	Trust model

Description

Interchain transactions can result in irretrievable locked funds if they fail on the ITS Hub. The infrastructure lacks fund recovery mechanisms. For example, hen a token is not deployed on the destination chain or issues occur during interchain token amount conversions, the transaction fails without the ability to recover the burned (or locked) tokens.

Exploit scenario

A user may lose funds if failures occur during the transaction relaying process:

1. Alice initiates an interchain transfer of 100 tokens from chain A to chain C via the ITS Hub.
2. The transaction succeeds on chain A, and Alice's tokens are burned on the source chain.
3. The transaction fails during relaying on the ITS Hub because the token contract has not been deployed on chain C.
4. The transaction is not successfully relayed to chain C, so no tokens are minted on the destination chain.
5. Since Alice's tokens were burned on chain A but not reminted on chain C, her funds are permanently lost and unrecoverable in the current implementation.

Recommendation

Enhance the ITS Hub to prevent irretrievable fund losses in interchain transactions that fail during relay on the ITS Hub. Implement recovery mechanisms or fallback procedures to mitigate the risk of irretrievable funds. If implementing comprehensive recovery mechanisms is not feasible, document these design limitations transparently.

Acknowledgment 1.1

The issue was acknowledged by the team with the comment:

> *This has been reported before, to address it we would need to introduce more vulnerabilities to prevent user errors that are not prevented in blockchain already (sending funds to a non-existent address causes loss of funds anywhere).*
>
> — Axelar Team

[Go back to Findings Summary]

L1: Empty Token Deployment in `InterchainTokenFactory.deployInterchainToken` Function

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	InterchainTokenFactory.sol	Type:	Data validation

Description

The `deployInterchainToken` function in the `InterchainTokenFactory` contract permits token deployment with both zero initial supply and no minter address. Deploying such a token is only a loss of funds for the deployer because the token cannot be used. ITS will only mint new tokens on interchain transactions, but because the token has no supply, no such transactions can be made.

Exploit scenario

1. Bob wants to deploy a new interchain token and calls `deployInterchainToken` without setting an initial supply or minter, assuming he will automatically be the minter.
2. The token is successfully deployed, but since it has no supply and no minting capabilities, it becomes unusable and only pollutes the state space.
3. Bob loses the full gas fees for the deployment instead of just the partial gas fees that would have occurred if the transaction had reverted.

Recommendation

Implement input validation to revert the transaction if both initial supply and minter address are zero.

Fix 1.1

The fix was implemented by adding an `else` branch to the `if` statement, which checks for initial supply and minter address:

Listing 1. Excerpt from InterchainTokenFactory

```
if (initialSupply > 0) {
    minterBytes = address(this).toBytes();
} else if (minter != address(0)) {
    if (minter == address(interchainTokenService)) revert
InvalidMinter(minter);

    minterBytes = minter.toBytes();
} else {
    revert ZeroSupplyToken();
```

Go back to Findings Summary

L2: Optional `ERC-20` Functions Required

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	InterchainTokenFactory.sol	Type:	Data validation

Description

The `InterchainTokenService` implementation assumes that canonical ERC-20 tokens registered in the service implement the following optional functions:

- `name()`
- `symbol()`
- `decimals()`

However, some `ERC-20` tokens may not implement these functions or may implement them with different return types than the ITS expects.

Exploit scenario

Alice registers the canonical `MKR` token and deploys a token manager for it. The `MKR` token returns `bytes32` for the `name` and `symbol` functions instead of the `string` type that the ITS expects. This type mismatch causes an ABI decoding revert.

Recommendation

Reevaluate support for non-standard tokens like `MKR`.

Acknowledgment 1.1

Acknowledged by the client.

> *There is no way to address this that fits the scope,*

> `InterchainTokenFactory` *is only a utility allowing people to register their tokens in a 'standard' way, if their tokens do not fit a 'standard' protocol (no fee-on-transfer, expose* `name()`, `symbol()`, `decimals()`*) then a custom token manager should be used instead.*
>
> — Axelar Team

Go back to Findings Summary

L3: Tokens with Callbacks Can Break Accounting

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	TokenHandler.sol	Type:	Logic error

Description

The `TokenHandler` contract implements the `_transferTokenFromWithFee` helper function that handles token transfers for tokens with on-transfer fees. The function returns the transferred amount excluding the fee, calculated as the difference in the sender's balance before and after the transaction. Although the `noReEntrancy` modifier prevents reentering this function via token callbacks, the sender's balance can be manipulated through direct transfers in the token callback. A malicious user can transfer tokens to themselves via a callback to artificially increase their balance. This vulnerability may lead to:

- breaking flow accounting (`flowIn` and `flowOut`);
- transferring the full token amount (including the fee) cross-chain.

Exploit scenario

Alice executes a cross-chain transfer. The ITS call invokes the `TokenHandler.takeToken` function, which calls `_transferTokenFromWithFee` and returns its value as the transfer amount. During the token callback, Alice can transfer tokens to herself to artificially increase her balance to make it look as if no fee was paid. This results in more tokens being transferred cross-chain than intended. The same scenario applies to the receiving side of the cross-chain transfer, where the full amount can be delivered to the receiving account through the `TokenHandler.giveToken` function.

Recommendation

Fixing the issue requires knowing the exact fee for the given token. Proposing a simple solution to the issue is thus quite hard. Therefore, reevaluate how important it is to support fee-on-transfer tokens.

Acknowledgment 1.1

The client acknowledged this issue, as fee-on-transfer tokens inherently cause accounting issues that cannot be fully mitigated in the protocol's design.

Go back to Findings Summary

W1: `GatewayCaller` Contract Ambiguously Reverts on Insufficient Funds

Impact:	Warning	Likelihood:	N/A
Target:	GatewayCaller.sol	Type:	Data validation

Description

The `GatewayCaller.callContract` function forwards `gasValue` amount of Ether to `gasService`, however it does not check if `msg.value` is at least of `gasValue`. When `msg.value` is lower than `gasValue` (and the contract lacks additional funds), the function reverts without providing an informative error message.

Recommendation

Implement a custom error (e.g., `NotEnoughFunds`) that triggers when the not sufficient funds are available to cover the `gasValue`.

Acknowledgment 1.1

Acknowledged by the client to maintain lower gas costs.

Go back to Findings Summary

W2: Missing Zero Address Check for Token in `TokenManagerProxy`

Impact:	Warning	Likelihood:	N/A
Target:	TokenManagerProxy.sol	Type:	Data validation

Description

When registering a canonical token as an interchain token, a token manager instance (`TokenManagerProxy`) is deployed for the token. The `TokenManagerProxy` constructor decodes the `tokenAddress` value from the `params` input parameter using the `TokenManager.getTokenAddressFromParams` function. The constructor, however, does not contain any check if `tokenAddress` value is non-zero. However, neither the constructor nor the entry point contracts (`InterchainTokenFactory` and `InterchainTokenService`) validate that this address is non-zero. This allows registering a canonical token with a zero address, which is not a valid token address.

Recommendation

The issue is closely tied with W3, which concerns not checking if the registered canonical token exists. Resolving that issue also simultaneously resolves this one. Alternatively, add a zero address check for the `tokenAddress` when deploying a new `TokenManagerProxy` proxy contract.

Fix 1.1

The issue was resolved by fixing issue W3.

Go back to Findings Summary

W3: Factory Does Not Check If Canonical Token Exists When Registering It

Impact:	Warning	Likelihood:	N/A
Target:	InterchainTokenFactory	Type:	Data validation

Description

The `InterchainTokenFactory.registerCanonicalInterchainToken` function does not verify the existence of the token being registered. Other factory functions, such as `deployRemoteCanonicalInterchainToken` and `deployRemoteInterchainToken`, include a check for token existence using the following code:

Listing 2. Excerpt from InterchainTokenFactory

```
// The 3 lines below will revert if the token does not exist.
string memory tokenName = token.name();
string memory tokenSymbol = token.symbol();
uint8 tokenDecimals = token.decimals();
```

This verification step is absent in the `registerCanonicalInterchainToken` function.

Recommendation

Add the missing check to the `registerCanonicalInterchainToken` function.

Fix 1.1

Fixed by adding the missing check using a new `_checkToken` function.

Go back to Findings Summary

W4: TokenManager Function Selector Clashes

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

Description

New token managers are deployed as the `TokenManagerProxy` contracts, which delegate-call to the pre-deployed `TokenManager` implementation contract. Both contracts include `address public immutable interchainTokenService;` in their bytecode. Thus, the selector in the proxy shadows the implementation selector. This shadowing also applies to other getters in `TokenManager`, namely:

- `tokenAddress`
- `interchainTokenId`
- `implementationType`.

These functions are implemented to revert when called from within the contract itself, as they are designed to be delegate-called by proxies only:

Listing 3. Excerpt from TokenManager

```
/**
 * @notice Reads the token address from the proxy.
 * @dev This function is not supported when directly called on the
implementation. It
 * must be called by the proxy.
 * @return tokenAddress_ The address of the token.
 */
function tokenAddress() external view virtual returns (address) {
    revert NotSupported();
}
```

In contrast, the `interchainTokenService` function does not follow this pattern and is fully implemented and callable from both the implementation and proxy contract. Although functional, mixing design approaches may be confusing for external viewers. Additionally, the `implementationType` function documentation does not mention that the function is intended to be used only from the implementation contract, while the other functions do.

Recommendation

Modify the `TokenManager.interchainTokenService` function to be a `pure` function which always reverts to unify with the other getter functions. Finally add explanation of the intent in the code documentation.

Partial solution 1.1

Code comments were added to clarify the intent of the functions.

Go back to Findings Summary

W5: Multiple Contracts Receive Ether Without Withdrawal Capability

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

Description

Several contracts in the codebase can receive Ether, but never send it.

The `TokenHandler` contract can receive Ether but never sends it. Even though Ether should not be sent to the contract under normal circumstances, in a rare case it would happen the Ether will be locked inside the contract without any means to retrieve it. The payable functions allowing this are:

- `takeToken`
- `postTokenManagerDeploy`.

The `TokenManager` contract can receive Ether via the `multicall` function which is `payable`. The contract, however, does not contain any function that sends ether, nor does the `TokenManagerProxy` which delegate-calls the `TokenManager` implementation.

Similarly, the `TokenManagerDeployer` contract marks the only `deployTokenManager` function as `payable` but the contract does not send ether.

Issues were detected using Wake static analysis.

Recommendation

Remove `payable` from functions in the mentioned contracts where possible.

Acknowledgment 1.1

The issue was acknowledged by the team with the comment:

> *The* `payable` *functions are intentionally designed to allow them to be delegate-called by other payable functions.*
>
> — Axelar Team

Go back to Findings Summary

I1: Implementation Details of `RolesBase` Exposed in `TokenManager`

Impact:	Info	Likelihood:	N/A
Target:	TokenManager.sol, RolesBase.sol	Type:	Code quality

Description

The `TokenManager` contract directly utilizes the bit shifting operation `1 << X`, which is an implementation detail of the `RolesBase` contract. This implementation detail appears in the `_addAccountRoles` and `_transferAccountRoles` functions, resulting in five instances of bit shifting operations throughout the `TokenManager` contract. The `RolesBase` contract could provide a more abstract interface through functions such as `_addRole` and `_transferRole` that accept multiple `uint8` role values, thereby encapsulating the bit shifting implementation and improving code clarity in `TokenManager`.

Recommendation

Improve the function interface for multiple roles to make the code clearer. Function overloading for `_addRole` and `_transferRole` functions could be used to add a variant taking two `uint8` role arguments, which would be sufficient in the current codebase to abstract the bit shifting.

Acknowledgment 1.1

The issue was acknowledged by the team with the comment:

> *There is a function to add multiple roles, which takes in* `uint8[]`*, but instantiating such an array is both more complicated in the code and uses more resources, so*

> *interacting with the implementation details of* `RolesBase` *was preferred.*
>
> — Axelar Team

Go back to Findings Summary

I2: Duplicate Message Type Definitions

Impact:	Info	Likelihood:	N/A
Target:	various	Type:	Unused code

Description

Message types are defined redundantly in two separate locations:

- as constants:

Listing 4. Excerpt from [InterchainTokenService]

```
uint256 private constant MESSAGE_TYPE_INTERCHAIN_TRANSFER = 0;
uint256 private constant MESSAGE_TYPE_DEPLOY_INTERCHAIN_TOKEN = 1;
uint256 private constant MESSAGE_TYPE_DEPLOY_TOKEN_MANAGER = 2;
uint256 private constant MESSAGE_TYPE_SEND_TO_HUB = 3;
uint256 private constant MESSAGE_TYPE_RECEIVE_FROM_HUB = 4;
```

- as enum:

Listing 5. Excerpt from [InterchainTokenServiceTypes]

```
enum MessageType {
    INTERCHAIN_TRANSFER,
    DEPLOY_INTERCHAIN_TOKEN,
    DEPLOY_TOKEN_MANAGER
}
```

Recommendation

Select one definition method and implement it consistently throughout the codebase. Remove the redundant definition.

Acknowledgment 1.1

The `MessageType` enum in the `InterchainTokenServiceTypes.sol` contract was extended with additional message types in the provided fix commit. However, this enum remains unused in the codebase.

Listing 6. Excerpt from [InterchainTokenServiceTypes]()

```
enum MessageType {
    INTERCHAIN_TRANSFER,
    DEPLOY_INTERCHAIN_TOKEN,
    DEPLOY_TOKEN_MANAGER,
    SEND_TO_HUB,
     RECEIVE_FROM_HUB
}
```

The client response:

> *InterchainTokenServiceTypes is not used in the actual code, not sure why it was added and it will likely be removed.*
>
> — Axelar Team

[Go back to Findings Summary]()

I3: Inconsistent Type Usage for `TokenManagerType`

Impact:	Info	Likelihood:	N/A
Target:	various	Type:	Code quality

Description

The `enum TokenManagerType` is inconsistently used with both `TokenManagerType` and `uint256` types throughout the codebase. Outside of the `InterchainTokenService.sol` contract, `uint256` is exclusively used, necessitating type casting when comparing values to enum constants.

Examples where `TokenManagerType` type is used:

Listing 7. Excerpt from InterchainTokenService

```
function deployTokenManager(
    bytes32 salt,
    string calldata destinationChain,
    TokenManagerType tokenManagerType,
    bytes calldata params,
    uint256 gasValue
) external payable whenNotPaused returns (bytes32 tokenId) {
```

Listing 8. Excerpt from InterchainTokenService

```
function _deployRemoteTokenManager(
    bytes32 tokenId,
    string calldata destinationChain,
    uint256 gasValue,
    TokenManagerType tokenManagerType,
    bytes calldata params
) internal {
```

Examples where `uint256` type is used:

Listing 9. Excerpt from [InterchainTokenService]

```
function tokenManagerImplementation(uint256 /*tokenManagerType*/) external
    view returns (address) {
```

Listing 10. Excerpt from [TokenHandler]

```
function postTokenManagerDeploy(uint256 tokenManagerType, address
    tokenManager) external payable {
```

Listing 11. Excerpt from [TokenManagerDeployer]

```
function deployTokenManager(
    bytes32 tokenId,
    uint256 implementationType,
    bytes calldata params
) external payable returns (address tokenManager) {
```

Recommendation

Standardize the type usage for `TokenManagerType`. Relocate the type definition to a dedicated `InterchainTokenServiceTypes.sol` file and implement it consistently throughout the codebase to improve explicitness and eliminate the need for type casting.

Acknowledgment 1.1

The issue was acknowledged by the team with the comment:

> *Noted, but changing function interfaces at this point seems more trouble than it's worth.*
>
> — Axelar Team

[Go back to Findings Summary]

I4: Excessive Inheritance in `InterchainToken`

Impact:	Info	Likelihood:	N/A
Target:	InterchainToken	Type:	Code quality

Description

On line 19, the `ERC20` in the inheritance list of `InterchainToken` is excessive because `InterchainToken` already extends `ERC20Permit`, which itself extends the `ERC20` contract.

Listing 12. Excerpt from [InterchainToken]()

```
19 contract InterchainToken is InterchainTokenStandard, ERC20, ERC20Permit,
   Minter, IInterchainToken {
```

Recommendation

Remove the excessive item `ERC20` from the inheritance list.

Acknowledgment 1.1

The team acknowledged the excessive `ERC20` inheritance because removing it would change the contract bytecode.

> *... Also removed some of the things that were addressed before, as they changed the bytecode of contracts that should have a constant bytecode, specifically* `InterchainToken` *and* `TokenManager`.
>
> — Axelar Team

[Go back to Findings Summary]()

I5: Typos or Missing Documentation

Impact:	Info	Likelihood:	N/A
Target:	various	Type:	Code quality

Description

Various files contain typos or missing documentation: - Documentation in `TokenHandler` mentions `This interface` instead of `This contract`:

Listing 13. Excerpt from [TokenHandler]

```
/**
 * @title TokenHandler
 * @notice This interface is responsible for handling tokens before
initiating an interchain token transfer, or after receiving one.
 */
contract TokenHandler is ITokenHandler, ITokenManagerType, ReentrancyGuard,
Create3AddressFixed {
```

- Constant in `ERC20Permit` is undocumented:

Listing 14. Excerpt from [ERC20Permit]

```
if (uint256(s) >
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) revert
InvalidS();
```

- `TokenManager` has a missing `@dev note` indicating that the intent is to only be called from TM proxy:

Listing 15. Excerpt from [TokenManager]

```
/**
 * @notice Returns implementation type of this token manager.
 * @return uint256 The implementation type of this token manager.
 */
function implementationType() external pure returns (uint256) {
    revert NotSupported();
}
```

Recommendation

Correct the typos and add the missing documentation.

Acknowledgment 1.1

Typos and missing documentation were addressed as follows:

- `TokenHandler` documentation was fixed;
- `ERC20Permit` missing documentation was acknowledged, because it changes the bytecode;
- `TokenManager` missing documentation was acknowledged, because it changes the bytecode.

> *... Also removed some of the things that were addressed before, as they changed the bytecode of contracts that should have a constant bytecode, specifically* `InterchainToken` *and* `TokenManager`.
>
> — Axelar Team

Go back to Findings Summary

I6: Function `TokenManager.tokenAddress` Can Be Marked as Pure

Impact:	Info	Likelihood:	N/A
Target:	TokenManager.sol	Type:	Code quality

Description

The `TokenManager.tokenAddress` function is marked as `view virtual`:

Listing 16. Excerpt from [TokenManager]

```
/**
 * @notice Reads the token address from the proxy.
 * @dev This function is not supported when directly called on the
implementation. It
 * must be called by the proxy.
 * @return tokenAddress_ The address of the token.
 */
function tokenAddress() external view virtual returns (address) {
    revert NotSupported();
}

/**
 * @notice A function that returns the token id.
 * @dev This will only work when implementation is called by a proxy, which
stores the tokenId as an immutable.
 * @return bytes32 The interchain token ID.
 */
function interchainTokenId() public pure returns (bytes32) {
    revert NotSupported();
}

/**
 * @notice Returns implementation type of this token manager.
 * @return uint256 The implementation type of this token manager.
 */
function implementationType() external pure returns (uint256) {
    revert NotSupported();
}
```

The function serves as a placeholder to be shadowed by the `TokenManagerProxy` contract, similar to the `interchainTokenId` and `implementationType` getter functions. These functions are implemented as `public immutable` variables in the `TokenManagerProxy` contract:

Listing 17. Excerpt from [TokenManagerProxy]()

```
contract TokenManagerProxy is BaseProxy, ITokenManagerProxy {
    bytes32 private constant CONTRACT_ID = keccak256('token-manager');

    address public immutable interchainTokenService;
    uint256 public immutable implementationType;
    bytes32 public immutable interchainTokenId;
    address public immutable tokenAddress;
```

Since these values are hard-coded during the `TokenManagerProxy` contract deployment, all three functions can be marked as `pure`.

Recommendation

Mark the `TokenManager.tokenAddress` function as `pure`, consistent with the `interchainTokenId` and `interchainTokenService` functions.

Acknowledgment 1.1

Acknowledged by the client:

> *We want to match the interface of ITokenManagerProxy.*
>
> — Axelar Team

[Go back to Findings Summary]()

I7: `InterchainTokenService.execute` Function Placement Inconsistency

Impact:	Info	Likelihood:	N/A
Target:	InterchainTokenService	Type:	Code quality

Description

The external `execute` function in the `InterchainTokenService` contract is incorrectly positioned within the internal functions section of the code.

Recommendation

Relocate the `execute` function from the internal functions section to the external functions section to maintain proper code organization.

Fix 1.1

Fixed by relocating the external `execute` function to the user functions section of the `InterchainTokenService` contract.

Go back to Findings Summary

I8: Unused Code

Impact:	Info	Likelihood:	N/A
Target:	various	Type:	Unused code

Description

The following errors are not used in the codebase:

Listing 18. Excerpt from ITokenManagerDeployer

```
error AddressZero();
```

Listing 19. Excerpt from ITokenManager

```
error TakeTokenFailed();
error GiveTokenFailed();
error NotToken(address caller);
error ZeroAddress();
error AlreadyFlowLimiter(address flowLimiter);
error NotFlowLimiter(address flowLimiter);
```

Listing 20. Excerpt from IInterchainTokenService

```
error InvalidTokenManagerImplementationType(address implementation);
```

Listing 21. Excerpt from IInterchainTokenFactory

```
error InvalidChainName();
```

Listing 22. Excerpt from IInterchainTokenFactory

```
error NotOperator(address operator);
error NotServiceOwner(address sender);
```

The following contracts are not used in `using-for` directives:

Listing 23. Excerpt from [TokenHandler]

```
using SafeTokenTransfer for IERC20;
```

Listing 24. Excerpt from [InterchainToken]

```
using AddressBytes for bytes;
```

Issues were detected using Wake static analysis.

Recommendation

Review all unused errors and `using-for` directives. Either implement them in the corresponding locations or remove them to simplify the codebase.

Fix 1.1

Fixed by removing unused legacy code.

Go back to Findings Summary

Report Revision 1.1

Revision Team

Revision team is the same as in Report Revision 1.0.

Since the previous revision 1.0, the following changes have been implemented:

- disallowed empty token deployment (tokens with both zero initial supply and no minter address);
- fixed missing token address check when registering canonical token; and
- various code cleanup improvements.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Axelar: Interchain Token Service, 16.1.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Fuzzing

The following table lists all implemented execution flows in the Wake fuzzing framework.

ID	Flow	Added
F1	Execution of interchain transfer	1.0

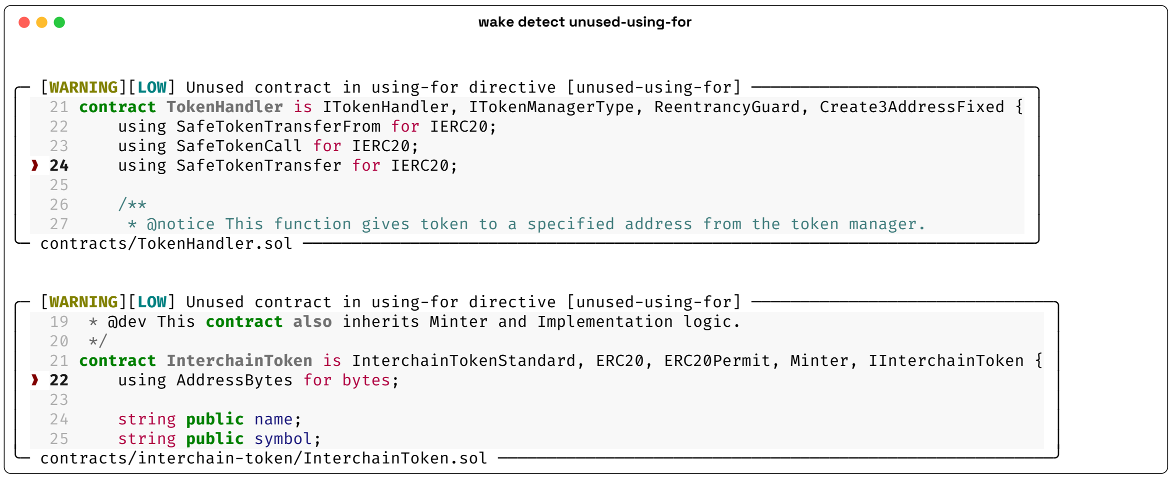
Table 4. Wake fuzzing flows

The following table lists the invariants checked after each flow.

ID	Invariant	Added	Status
IV1	Token balances are correct accross connected chains	1.0	Success

Table 5. Wake fuzzing invariants

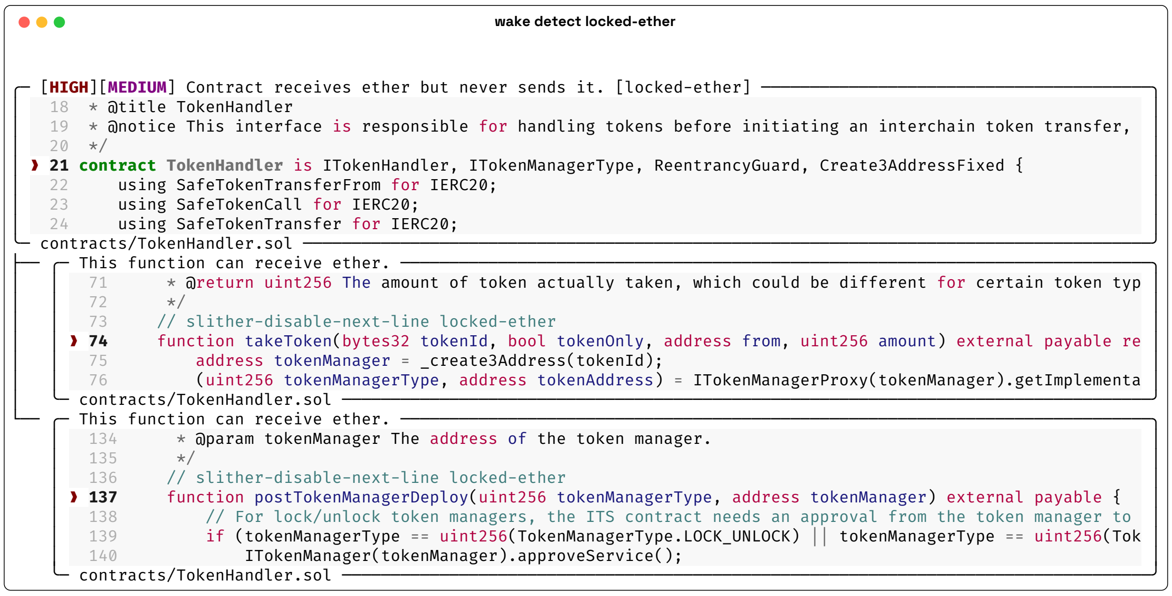
B.2. Detectors


wake detect unused-using-for

```
╭─ [WARNING][LOW] Unused contract in using-for directive [unused-using-for] ─────────────────────────────╮
│  21 contract TokenHandler is ITokenHandler, ITokenManagerType, ReentrancyGuard, Create3AddressFixed {  │
│  22     using SafeTokenTransferFrom for IERC20;                                                          │
│  23     using SafeTokenCall for IERC20;                                                                  │
│ ❱ 24     using SafeTokenTransfer for IERC20;                                                             │
│  25                                                                                                      │
│  26     /**                                                                                              │
│  27      * @notice This function gives token to a specified address from the token manager.            │
╰─ contracts/TokenHandler.sol ───────────────────────────────────────────────────────────────────────────╯

╭─ [WARNING][LOW] Unused contract in using-for directive [unused-using-for] ───────────────────────────────╮
│  19  * @dev This contract also inherits Minter and Implementation logic.                                 │
│  20  */                                                                                                   │
│  21 contract InterchainToken is InterchainTokenStandard, ERC20, ERC20Permit, Minter, IInterchainToken {  │
│ ❱ 22     using AddressBytes for bytes;                                                                    │
│  23                                                                                                       │
│  24     string public name;                                                                               │
│  25     string public symbol;                                                                             │
╰─ contracts/interchain-token/InterchainToken.sol ─────────────────────────────────────────────────────────╯
```

Figure 1. Unused using for *detector*


wake detect locked-ether

```
╭─ [HIGH][MEDIUM] Contract receives ether but never sends it. [locked-ether] ──────────────────────────────────────╮
│  18  * @title TokenHandler                                                                                       │
│  19  * @notice This interface is responsible for handling tokens before initiating an interchain token transfer, │
│  20  */                                                                                                          │
│ ❱ 21 contract TokenHandler is ITokenHandler, ITokenManagerType, ReentrancyGuard, Create3AddressFixed {          │
│  22     using SafeTokenTransferFrom for IERC20;                                                                  │
│  23     using SafeTokenCall for IERC20;                                                                          │
│  24     using SafeTokenTransfer for IERC20;                                                                      │
╰─ contracts/TokenHandler.sol ─────────────────────────────────────────────────────────────────────────────────────╯
├── ╭─ This function can receive ether. ─────────────────────────────────────────────────────────────────────────╮
│   │  71      * @return uint256 The amount of token actually taken, which could be different for certain token typ │
│   │  72      */                                                                                                  │
│   │  73     // slither-disable-next-line locked-ether                                                           │
│   │ ❱ 74     function takeToken(bytes32 tokenId, bool tokenOnly, address from, uint256 amount) external payable re │
│   │  75         address tokenManager = _create3Address(tokenId);                                                 │
│   │  76         (uint256 tokenManagerType, address tokenAddress) = ITokenManagerProxy(tokenManager).getImplementa │
│   ╰─ contracts/TokenHandler.sol ───────────────────────────────────────────────────────────────────────────────╯
└── ╭─ This function can receive ether. ─────────────────────────────────────────────────────────────────────────╮
    │  134      * @param tokenManager The address of the token manager.                                           │
    │  135      */                                                                                                │
    │  136     // slither-disable-next-line locked-ether                                                          │
    │ ❱ 137     function postTokenManagerDeploy(uint256 tokenManagerType, address tokenManager) external payable { │
    │  138         // For lock/unlock token managers, the ITS contract needs an approval from the token manager to │
    │  139         if (tokenManagerType == uint256(TokenManagerType.LOCK_UNLOCK) || tokenManagerType == uint256(Tok │
    │  140             ITokenManager(tokenManager).approveService();                                              │
    ╰─ contracts/TokenHandler.sol ───────────────────────────────────────────────────────────────────────────────╯
```

Figure 2. Sample from Locked ether *detector*

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz