Audited by CRED SHIELDS

CredShields

Smart Contract Audit

February 4th, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Landslide between January 3rd, 2025, and January 22nd, 2025. A retest was performed on January 31st, 2025.

Author

Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor)

Prepared for

Landslide

Table of Contents

1. Executive Summary ----------

Landslide engaged CredShields to perform a smart contract audit from January 3rd, 2025, to January 22nd, 2025. During this timeframe, 23 vulnerabilities were identified. **A retest was performed on January 31st, 2025, and all the bugs have been addressed.**

During the audit, 7 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Landslide" and should be prioritized for remediation.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
ICM Contracts	4	3	1	4	2	9	**23**
	4	**3**	**1**	**4**	**2**	**9**	**23**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the ICM Contract's scope during the testing window while abiding by the policies set forth by Landslide's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Landslide's internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Landslide can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Landslide can future-proof its security posture and protect its assets.

2. The Methodology ----------

Landslide engaged CredShields to perform the ICM Contracts Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation Phase

The CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from January 3rd, 2025, to January 22nd, 2025, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
https://github.com/LandslideNetwork/icm-contracts/tree/9827b2af0cc27eea0345ede1c542277a4230382d

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting Phase

Landslide is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. **Informational**

   We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- Shashank, Co-founder CredShields shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary ----------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 23 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Token is transferred to the relayer instead of recipient	Critical	Business Logic Issue
Missing msg.sender validation in transfer() function	Critical	Business Logic Issue
Missing token burning feature in _processFromCosmosSend() function	Critical	Business Logic Issue
Incorrect input parameter is passed while transferring message in _toCosmosSend()	Critical	Business Logic Issue
Low valued token in primaryFeeTokenAddress may cause users to bypass fee mechanism	High	Missing Input Validation
Lack of primaryFee amount validation can lead to bypassing fee mechanism	High	Missing Input Validation
Multi-hop cosmos transfer will always revert	High	Missing Implementation

Frontrunning issue while registering with home contract	Medium	Frontrunning
Use Ownable2Step	Low	Missing Best Practices
Missing Events in Important Functions	Low	Missing Best Practices
Missing Zero Address Validations	Low	Missing Input Validation
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Hardcoded Static Address	Informational	Missing Best Practices
Dead Code	Informational	Code With No Effects (SWC-135)
Large Number Literals	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas optimization
Custom error to save gas	Gas	Gas Optimization
Cheaper Inequalities in if()	Gas	Gas Optimization
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Cheaper Inequalities in require()	Gas	Gas Optimization
Public Constants can be Private	Gas	Gas Optimization
Splitting Require/Revert Statements	Gas	Gas Optimization

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use

SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found
SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.

SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status ----------

Landslide is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on January 31st, 2025, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Token is transferred to the relayer instead of recipient	Critical	**Fixed** [Jan 31, 2025]
Missing msg.sender validation in transfer() function	Critical	**Not Applicable** [Jan 31, 2025]
Missing token burning feature in _processFromCosmosSend() function	Critical	**Not Applicable** [Jan 31, 2025]
Incorrect input parameter is passed while transferring message in _toCosmosSend()	Critical	**Fixed** [Jan 31, 2025]
Low valued token in primaryFeeTokenAddress may cause users to bypass fee mechanism	High	**Not Fixed** [Jan 31, 2025]
Lack of primaryFee amount validation can lead to bypassing fee mechanism	High	**Not Fixed** [Jan 31, 2025]
Multi-hop cosmos transfer will always revert	High	**Not Applicable** [Jan 31, 2025]
Frontrunning issue while registering with home contract	Medium	**Fixed** [Jan 31, 2025]
Use Ownable2Step	Low	**Fixed** [Jan 31, 2025]
Missing Events in Important Functions	Low	**Fixed** [Jan 31, 2025]
Missing Zero Address Validations	Low	**Partially Fixed** [Jan 31, 2025]

Floating and Outdated Pragma	Low	**Partially Fixed** [Jan 31, 2025]
Hardcoded Static Address	Informational	**Fixed** [Jan 31, 2025]
Dead Code	Informational	**Not Applicable** [Jan 31, 2025]
Large Number Literals	Gas	**Not Applicable** [Jan 31, 2025]
Gas Optimization in Increments	Gas	**Not Applicable** [Jan 31, 2025]
Custom error to save gas	Gas	**Not Applicable** [Jan 31, 2025]
Cheaper Inequalities in if()	Gas	**Not Applicable** [Jan 31, 2025]
Gas Optimization in Require/Revert Statements	Gas	**Not Applicable** [Jan 31, 2025]
Cheaper Conditional Operators	Gas	**Not Applicable** [Jan 31, 2025]
Cheaper Inequalities in require()	Gas	**Not Applicable** [Jan 31, 2025]
Public Constants can be Private	Gas	**Not Applicable** [Jan 31, 2025]
Splitting Require/Revert Statements	Gas	**Not Applicable** [Jan 31, 2025]

Table: Summary of findings and status of remediation

5. Bug Reports ----------

Bug ID #1 [Fixed]

Token is transferred to the relayer instead of recipient

Vulnerability Type
Business Logic Issue

Severity
Critical

Description
The _toCosmosWithdraw() function in the TokenRemote.sol contract contains a flaw in how it handles token withdrawal and distribution. Specifically, when the TO_COSMOS_SINGLE_HOP_SEND message is received, the _toCosmosWithdraw() function is invoked via the _receiveTeleporterMessage() mechanism, the relayer (teleporter contract) becomes the msg.sender. In the _toCosmosWithdraw() function, the wrapped token is burned from the msg.sender, and the corresponding native token is sent back to msg.sender. This results in the relayer receiving the native tokens instead of the intended recipient. And theTokensWithdrawn event is emitted for the recipient.
The primary purpose of this function is to burn the wrapped tokens and send the native tokens to the recipient specified in the input data. However, due to the misuse of msg.sender, the funds are incorrectly directed to the relayer, creating a critical issue in the token bridging mechanism.

Affected Code
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L308

Impacts
The intended recipient does not receive the native tokens. Instead, the relayer is rewarded with the native tokens, making the fund loss for this recipient.

Remediation

Modify the _toCosmosWithdraw() function to ensure that the native tokens are sent to the recipient specified in the function's input. Replace _msgSender() with recipientAddress in both the _burn and sendValue operations.

Retest
This issue has been resolved by removing the logic of _toCosmosWithdraw()

Bug ID #2 [Not Applicable]

Missing msg.sender validation in transfer() function

Vulnerability Type
Business Logic Issue

Severity
Critical

Description
The _processFromCosmosSend() function does not validate the msg.sender before executing the transfer logic. Proper validation of msg.sender is essential in a bridging system to ensure that tokens are locked or burned from the correct account on the source chain. Without this validation, any user can craft and submit transactions that lead to unauthorized token bridging.
Since the function lacks checks to verify that msg.sender is the rightful owner of the tokens being transferred, a malicious actor can use this flaw to bypass ownership requirements. This creates a significant risk of unauthorized bridging, where the protocol mints tokens on the destination blockchain without locking or burning tokens from the legitimate user's account on the source chain.

Affected Code
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L69-L88

Impacts
This vulnerability allows attackers to exploit the protocol by initiating token transfers without holding or locking the corresponding amount of tokens in their own account. As a result, the protocol mints tokens on the destination chain, creating an imbalance in the token supply across chains and resulting in financial losses for the protocol.

Remediation
Add strict validation for msg.sender to ensure that the function is only executed by the account holding the tokens to be locked or burned. Before initiating the bridging process, verify that msg.sender has sufficient token balance and permission to execute the transaction.

Retest
Client's Comment: msg.sender is validated inside PacketSender.sendPacket function. This is precompile function in Landslide EVM and audited by Halborn team

Bug ID #3 [Not Applicable]

Missing token burning feature in _processFromCosmosSend() function

Vulnerability Type
Business Logic Issue

Severity
Critical

Description
The _processFromCosmosSend() function in the remote contract does not implement a token locking or burning mechanism before initiating the transfer to the destination blockchain. In contrast, the _processSend() function properly calls _prepareSend(), which ensures that tokens are burned on the source chain, maintaining the total supply integrity. The absence of this validation in _processFromCosmosSend() allows users to send tokens from the remote chain without locking or burning the corresponding tokens. As a result, the destination blockchain mints the desired tokens for the user without guaranteeing that the source tokens are secured, effectively leading to an uncontrolled increase in token supply on the destination chain.

Affected Code
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L572-L614

Impacts
This allows malicious actors to mint tokens on the destination blockchain without locking or burning them in the source chain.

Remediation
Implement a token locking or burning mechanism in the _processFromCosmosSend() function. This mechanism should ensure that tokens on the source chain are locked or burned before initiating the transfer to the destination blockchain.

Retest
Client's Comment: We don't generate/burn tokens in our subnet when we transfer from/to cosmos chain. so, there is nothing to burn.

Bug ID #4 [Fixed]

Incorrect input parameter is passed while transferring message in _toCosmosSend()

Vulnerability Type
Business Logic Issue

Severity
Critical

Description
The _toCosmosSend() function uses an incorrect parameter when preparing the TransferrerMessage. Specifically, instead of using the input.destinationCosmosBlockchainID parameter, the code mistakenly references input.destinationBlockchainID. This mismatch can result in the creation of a message containing an invalid or unintended blockchain ID, which may lead to miscommunication between systems or failed transactions.
The destinationCosmosBlockchainID parameter is critical for ensuring that the message is routed to the appropriate Cosmos blockchain recipient. By substituting this parameter with destinationBlockchainID, the function risks transmitting incorrect data, which may disrupt the intended functionality of the transfer.

Affected Code
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L318

Impacts
The incorrect use of destinationBlockchainID in place of destinationCosmosBlockchainID can cause messages to be sent to the wrong blockchain destination. This would result in failed or misrouted transfers.

Remediation
To resolve this issue, the function should be updated to correctly reference the destinationCosmosBlockchainID parameter when creating the TransferrerMessage.

Retest
This issue has been fixed by updating the parameters as recommended.

Bug ID #5 [Not Applicable]

Low valued token in primaryFeeTokenAddress may cause users to bypass fee mechanism

Vulnerability Type
Missing Input Validation

Severity
High

Description
The functions send(), toCosmosSend(), fromCosmosSend() and sendAndCall() within the smart contract accept an input parameter, primaryFeeTokenAddress. This lack of validation allows users to specify a low-value or non-standard token as the primary fee token. Consequently, this could undermine the fee mechanism and create an opportunity for attackers to bypass fees altogether.
By design, these functions rely on user-supplied input for fee payments, assuming the integrity of the provided token address. However, without explicit whitelisting or validation, attackers can supply arbitrary token addresses that are not aligned with the expected fee requirements, exploiting the lack of control and impacting the economic assumptions of the contract.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L253
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L307
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L388
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L416
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L498
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L540
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L603
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L630

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L719
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L746

Impacts

The absence of validation for the primaryFeeTokenAddress parameter allows users to bypass the intended fee mechanism by specifying low-value tokens as the fee. This enables users to pay negligible or no fees while still executing transactions. As a result, the contract effectively incurs a loss, as it accepts tokens with little to no value as payment for the required fees.

Remediation

To address this vulnerability, a whitelisting mechanism should be implemented to validate the primaryFeeTokenAddress input parameter.

Retest

Client's Comment: This is designed initially by Avalabs and we raised a question in their github issue: https://github.com/ava-labs/icm-contracts/issues/706

Bug ID #6 [Not Applicable]

Lack of primaryFee amount validation can lead to bypassing fee mechanism

Vulnerability Type
Missing Input Validation

Severity
High

Description
The functions send(), toCosmosSend(), fromCosmosSend() and sendAndCall() within the smart accept the primaryFee parameter without validating its value. This lack of validation allows users to specify 0 as the fee amount, effectively bypassing the fee mechanism entirely. The current implementation does not distinguish between sponsored and non-sponsored transactions, nor does it enforce any minimum fee requirement or dynamic fee calculation.

This issue arises because the smart contract does not verify whether the provided fee is appropriate for the transaction context. Consequently, users can exploit this weakness to avoid paying fees entirely, undermining the integrity of the fee system and disrupting its intended operation.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L254C23-L254C40
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L308
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L365
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L417
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L466
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L541
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L604
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L631
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L693

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L747

Impacts

The absence of validation for the primaryFee parameter enables users to execute transactions without paying the required fees by setting the fee amount to zero.

Remediation

To address this vulnerability, the contract should implement a mechanism to validate the primaryFee parameter. For non-sponsored transactions, the contract should enforce a minimum fee threshold or calculate fees dynamically based on transaction complexity and gas usage. If the transaction is sponsored, a flag or condition should be checked to ensure the sponsor covers the required fee. This dual approach will prevent users from circumventing the fee mechanism while allowing legitimate sponsored transactions to proceed as intended.

Retest

Client's Comment: This was designed initially by Avalabs and we raised a question in their github issue: https://github.com/ava-labs/icm-contracts/issues/706

Bug ID #7 [Not Applicable]

Multi-hop cosmos transfer will always revert

Vulnerability Type
Missing Implementation

Severity
High

Description
The _fromCosmosSend() function delegates multi-hop transfers to the _processFromCosmosSendMultiHop() function when the input.destinationBlockchainID does not match the _tokenHomeBlockchainID. However, the _processFromCosmosSendMultiHop function lacks a proper implementation and immediately reverts with a generic error message, "TokenRemote: invalid destination blockchain ID." This results in the inability to handle valid multi-hop transfers, leading to failed transactions in scenarios where multi-hop functionality is required.
Multi-hop transfers are essential for scenarios where tokens need to traverse multiple blockchain networks before reaching their final destination. The absence of a proper implementation for _processFromCosmosSendMultiHop prevents the system from supporting such transfers, limiting its utility and potentially frustrating users who rely on this feature.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L672-L677

Impacts
The missing implementation in _processFromCosmosSendMultiHop causes all valid multi-hop transfer attempts to fail with a revert error.

Remediation
To address this issue, a comprehensive implementation for the _processFromCosmosSendMultiHop function must be added. This implementation should handle the logic required to facilitate multi-hop transfers, including verifying destination parameters, processing fees, and forwarding the tokens to the next blockchain in the route.

Retest

If you don't support multi-hop, then consider removing the functionality as it will make code more readable.

Client's Comment: We don't support multi-hop on Avalanche L1 chains. There are only Avalanche C-Chain as a primary network and Landslide EVM L1 as a subnet. Token path is C-Chain → Landslide EVM L1 → Cosmos chain

Bug ID #8 [Fixed]

Frontrunning issue while registering with home contract

Vulnerability Type
Frontrunning

Severity
Medium

Description
The `registerWithHome()` function in the `TokenRemote.sol` contract allows the remote contract to be registered with its corresponding home contract. However, this function is vulnerable to frontrunning attacks due to the lack of access control mechanisms. A malicious actor can monitor pending transactions and preemptively execute the `registerWithHome()` function using different input parameters, effectively hijacking the registration process.

This issue arises because there is no mechanism to verify that the caller of the `registerWithHome()` function is authorized to perform the registration. As a result, the malicious actor can manipulate the function's behavior by providing their own parameters, which could cause the protocol to incorrectly register an unauthorized or invalid remote contract.

Affected Code
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L211-L237

Impacts
The malicious actor can alter the TeleporterFeeInfo params of the teleporter.

Remediation
Introduce an access control mechanism to restrict the execution of the registerWithHome() function to authorized addresses, such as the deployer or a pre-approved operator of the remote contract.

Retest
This issue has been fixed by added onlyOwner() modifier in the function.

Bug ID #9 [Fixed]

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Low

Description
The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ERC20MintBurnToken.sol#L8
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L107
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L43

Impacts
Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation
It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:
This issue has been fixed as recommended.

Bug ID #10 [Fixed]

Missing Events in Important Functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L762-L774
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L786-L805
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L818-L839
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L83-L89
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L91-L96
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L61-L63
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L65-L67
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L69-L88

Impacts
Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation
Consider emitting events for important functions to keep track of them.

Retest
This issue has been fixed as recommended.

Bug ID #11 [Partially Fixed]

Missing Zero Address Validations

Vulnerability Type
Missing Input Validation

Severity
Low

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L57
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L58

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This issue has been partially fixed.

Bug ID #12 [Partially fixed]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SWC-103)

Severity
Low

Description
Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract allowed floating or unlocked pragma to be used, i.e., ^0.8.25, 0.8.15. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCApp.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/interfaces/IIBC.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/interfaces/IERC20TokenHome.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/interfaces/INativeTokenHome.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/interfaces/ITokenHome.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/ERC20TokenHome.sol#L6

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/ERC20TokenHomeUpgradeable.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/NativeTokenHome.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/NativeTokenHomeUpgradeable.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/interfaces/INativeTokenRemote.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/interfaces/ITokenRemote.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemote.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemote.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L6
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ERC20MintBurnToken.sol#L2
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/WrappedNativeToken.sol#L6

Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.

Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.

The likelihood of exploitation is low.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.28 pragma version

Reference: https://swcregistry.io/docs/SWC-103

Retest

This issue has been partially fixed. The floating pragma version has been replaced with the fixed version. However, the pragma version is 0.8.25. It is recommended to use 0.8.28

Bug ID #13 [Fixed]

Hardcoded Static Address

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
The contract was found to be using hardcoded addresses.
This could have been optimized using dynamic address update techniques along with proper access control to aid in address upgrade at a later stage.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L142
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L100
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L109
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L118
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L124
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L177

Impacts
Hardcoding address variables in the contract make it difficult for it to be modified at a later stage in the contract as everything will need to be deployed again at a different address if there's a code upgrade.

Remediation
It is recommended to create dynamic functions to address upgrades so that it becomes easier for developers to make changes at a later stage if necessary.

The said function should have proper access controls to make sure only administrators can call that function using access control modifiers.
There should also be a zero address validation in the function to make sure the tokens are not lost.
If the address is supposed to be hardcoded, it is advisable to make it a constant if its value is not getting updated.

Retest
This has been fixed.

Bug ID#14 [Not Applicable]

Dead Code

Vulnerability Type
Code With No Effects - SWC-135

Severity
Informational

Description
It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities. The functions and parameters, contracts, and interfaces that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20BankTransferApp.sol#L50

Impacts
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement.
This reduces the overall size of the contracts and also helps in saving gas.

Remediation
If the library functions are not supposed to be used anywhere, consider removing them from the contract.

Retest
-

Bug ID #15 [Not Applicable]

Large Number Literals

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L180

Impacts
Having a large number literals in the code increases the gas usage of the contract during its deployment and when the functions are used or called from the contract.
It also makes the code harder to read and audit and increases the chances of introducing code errors.

Remediation
Scientific notation in the form of 2e10 is also supported, where the mantissa can be fractional, but the exponent has to be an integer. The literal MeE is equivalent to M * 10**E. Examples include 2e10, 2e10, 2e-10, 2.5e1, as suggested in official solidity documentation. https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals
It is recommended to use numbers in the form "35 * 1e7 * 1e18" or "35 * 1e25".
The numbers can also be represented by using underscores between them to make them more readable such as "35_00_00_000"

Retest
-

Bug ID#16 [Not Applicable]

Gas Optimization in Increments

Vulnerability Type
Gas optimization

Severity
Gas

Description
The contract uses two for loops**,** which use post increments for the variable "**i**".
The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L216

Impacts
Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation
It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and meanwhile saves some gas.

Retest

Bug ID #17 [Not Applicable]

Custom error to save gas

Vulnerability Type
Gas Optimization

Severity
Gas

Description
During code analysis, it was observed that the smart contract is using the revert() statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional revert(). Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the revert() statements.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L167

Impacts
Custom errors allow developers to provide more descriptive error messages with dynamic data. This provides better insights into the cause of the error, making it easier for users and developers to understand and address issues.

Remediation
It is recommended to replace all the instances of revert() statements with error() to save gas..

Retest
-

Bug ID #18 [Not Applicable]

Cheaper Inequalities in if()

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be doing comparisons using inequalities inside the "if" statement. When inside the "if" statements, non-strict inequalities (>=, <=) are usually cheaper than the strict equalities (>, <).

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L547
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L856
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L900
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L940
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/ERC20TokenHomeUpgradeable.sol#L261
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L355
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L386
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L241
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L303

Impacts
Using strict inequalities inside "if" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the strict inequalities with the non-strict ones to save gas as long as the logic of the code is not affected.

Retest:
-

Bug ID #19 [Not Applicable]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require/revert** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L136-139
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L180-180
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L181-183
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L184-187
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L188-191
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L192-195
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L196-198
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L247-247
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L301-301
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L410-410
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L522-522

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L604-607
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L608-611
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L826-826
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L861-861
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L894-894
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L934-934
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L963-963
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L999-999
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1000-1000
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1001-1001
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1002-1005
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1006-1006
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1007-1007
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1013-1013
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1019-1019
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1011-1011
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1012-1012
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1018-1018
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1017-1017
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L39-39
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L40-43

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L46-46
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L54-54
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L59-59
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L64-64
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L73-73
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L82-82
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L87-87
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L98-98
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L43-43
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L63-63
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L110-110
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L177-177
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L171-171
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L182-182
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L231-234
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L250-255
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L167-167
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L178-181

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L182-185
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L186-186
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L187-190
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L191-194
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L310-310
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L356-359
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L360-362
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L409-409
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L516-522
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L676-676
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L800-803
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L804-804
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L805-805
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L823-823
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L818-821
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L827-827
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L853-853
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L828-828
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L854-854
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L829-832
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L855-858

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L866-869
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L833-836
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L859-862
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L847-847
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L848-851
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L870-873
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L874-876
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L877-877
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L878-878
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L879-882
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L883-885

Impacts
Having longer require/revert strings than **32 bytes** cost a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require/revert** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest
-

Bug ID #20 [Not Applicable]

Cheaper Conditional Operators

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L522-L522
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L837-L837
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L994-L994
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L910-L910
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L950-L950
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1000-L1000
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1001-L1001
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1012-L1012
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1018-L1018
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L1017-L1017

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L547-L547
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L856-L856
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L900-L900
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L940-L940
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/ERC20TokenHomeUpgradeable.sol#L261-L261
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L251-L253
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L241-L241
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L303-L303
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L310-L310
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L828-L828
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L854-L854
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L847-L847
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L877-L877
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L878-L878

Impacts

Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Remediation

Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest

-

Bug ID #21 [Not Applicable]

Cheaper Inequalities in require()

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L137-L137
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L193-L193
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L963-L963
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L46-L46
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L98-L98
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/IBCBaseFungibleTokenApp.sol#L177-L177
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L188-L188
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L192-L192

Impacts
Using non-strict inequalities inside “require” statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the non-strict inequalities with the strict ones to save gas as long as the logic of the code is not affected.

Retest:

-

Bug ID #22 [Not Applicable]

Public Constants can be Private

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/TokenHome.sol#L95-L96
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenHome/ERC20TokenHomeUpgradeable.sol#L52-L53
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L122-L122
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L21-L21
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/ICS20Bank.sol#L22-L22
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L77-L78
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L100-L100
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L109-L109
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L118-L118

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/NativeTokenRemoteUpgradeable.sol#L123-L124
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/ERC20TokenRemoteUpgradeable.sol#L55-L56
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L109-L110
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L126-L126
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L134-L134
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L140-L140
- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/TokenRemote/TokenRemote.sol#L145-L145

Impacts
Public constants are more costly due to the default getter functions created for them, increasing the overall gas cost.

Remediation
If reading the values for the constants is not necessary, consider changing the public visibility to private.

Retest
-

Bug ID #23 [Not Applicable]

Splitting Require/Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Require/Revert statements when combined using operators in a single statement usually lead to a larger deployment gas cost but with each runtime calls, the whole thing ends up being cheaper by some gas units.

Affected Code

- https://github.com/LandslideNetwork/icm-contracts/blob/9827b2af0cc27eea0345ede1c542277a4230382d/contracts/ictt/ICS20/TokenRouter.sol#L209-L214

Impacts
The multiple conditions in one **require/revert** statement combine require/revert statements in a single line, increasing deployment costs and hindering code readability.

Remediation
It is recommended to separate the **require/revert** statements with one statement/validation per line.

Retest
-

6. The Disclosure ----------

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.

YOUR SECURE FUTURE STARTS HERE

CRED SHIELDS

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

Audited by CRED SHIELDS

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com