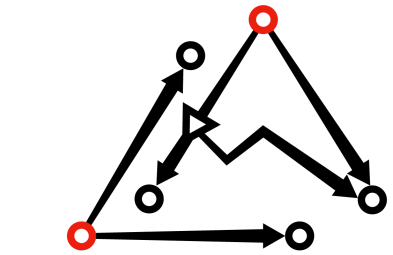

Least Authority

PRIVACY MATTERS

Smart Contracts

Security Audit Report

Hilter

Final Audit Report: 19 July 2025

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
    - Issue A: Missing Check for the Distributor Address Can Lead to the Loss of Access Control for the Respective Token
    - Issue B: ETH Can Be Locked in the Contract Indefinitely
    - Issue C: Updates Between Non-Zero Allowances Can Result in Exploits
  - Suggestions
    - Suggestion 1: Add Check for Out-Of-Bound Values in getImplementation
    - Suggestion 2: Add Check To Validate the Parameters of addTrustedAddress
    - Suggestion 3: Resolve TODOs in Codebase
    - Suggestion 4: Enable Two-Step Ownership Transfers
    - Suggestion 5: Update Implementation To Make It Consistent With the Specification Mentioned in the Comment
    - Suggestion 6: Remove Unnecessary Modifier
- About Least Authority
- Our Methodology

Overview

Background

Hilter has requested that Least Authority perform a security audit of their Smart Contracts.

Project Dates

- **June 21, 2025 - July 5, 2025:** Initial Code Review *(Completed)*
- **July 7, 2025:** Delivery of Initial Audit Report *(Completed)*
- **July 17:** Verification Review *(Completed)*
- **July 19:** Delivery of Final Audit Report *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer
- Mukesh Jaiswal, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Smart Contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- `Interchain-token-service`: https://gitlab.com/hilterltd-group/interchain-token-service
    - Specifically: https://gitlab.com/hilterltd-group/interchain-token-service/tree/feat/get-chain-name

Specifically, we examined the Git revisions for our initial review:

- Initial commit: `f719f44bd278ba26ff38f30664aedc766a37f586`
- Updated commit: `6f7b4eb3d4aaab9d8fe6ad726a3d9a86446ce902`

For the verification, we examined the Git revision:

- `cab1bb3b3e220a368bc932ba7ff9dfb5f5e8c872`

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Hilter:
https://hilter.com

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use of the smart contracts or disrupt their execution;
- Vulnerabilities in the smart contracts' code;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security review of the Hilter Smart Contracts, specifically the Interchain Token Service functionality. Our team found that the Hilter team has taken security into consideration in the design and implementation as demonstrated by appropriate permissions on sensitive functions, a failsafe for pausing the system, and attention to details, such as using the `SafeTransfer` token methods. However, our team identified implementation Issues that can lead to security vulnerabilities, as well as suggestions that include adherence to best practice recommendations and efficiency optimizations.

Code Quality

Our team performed a manual review of the smart contracts and found the code to be very well-organized and utilizing interfaces and inheritance efficiently. The codebase adheres to Solidity standards and best practices by implementing modern error handling, appropriate re-entrancy safeguards, and gas optimizations.

Tests

The Hilter Smart Contracts are sufficiently tested.

Documentation

The project documentation available for this review was sufficient and provided an accurate description of the protocol. Furthermore, every function in the codebase is well-commented according to NatSpec guidelines and includes relevant descriptions that facilitate understanding the code.

Scope

The scope of this review was sufficient and included all security-critical components.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Missing Check for the Distributor Address Can Lead to the Loss of Access Control for the Respective Token	Resolved
Issue B: ETH Can Be Locked in the Contract Indefinitely	Resolved
Issue C: Updates Between Non-Zero Allowances Can Result in Exploits	Resolved
Suggestion 1: Add Check for Out-Of-Bond Values in getImplementation	Resolved
Suggestion 2: Add Check To Validate the Parameters of addTrustedAddress	Resolved
Suggestion 3: Resolve TODOs in Codebase	Partially resolved
Suggestion 4: Enable Two-Step Ownership Transfers	Resolved
Suggestion 5: Update Implementation To Make It Consistent With the Specification Mentioned in the Comment	Partially resolved
Suggestion 6: Remove Unnecessary Modifier	Resolved

Issue A: Missing Check for the Distributor Address Can Lead to the Loss of Access Control for the Respective Token

Location

`contracts/interchain-token-service/InterchainTokenService.sol#L612`

Synopsis

In the function `_processDeployStandardizedTokenAndManagerPayload`, the distributor address is obtained, in bytes, through the function `distributorBytes`, after which it is converted to `address`. There is no check verifying that it is not `address(0)`. Only the `distributorBytes` length is checked, which is not sufficient because the following address can be passed:

`address(0) {"0x0000000000000000000000000000000000000000"}`

Impact

If the function `distributorBytes` is an `address(0)`, then access control for that token can be lost.

Preconditions

This Issue is possible if the value of `distributorBytes` is:

`0x0000000000000000000000000000000000000000`.

Feasibility

Low.

Remediation

We recommend adding an `address(0)` check after obtaining the distributor address from the function `distributerBytes`.

Verification

Resolved

Issue B: ETH Can Be Locked in the Contract Indefinitely

Location

`contracts/utils/StandardizedTokenDeployer.sol#L45-L54`

`contracts/utils/TokenManagerDeployer.sol#L33`

Synopsis

These functions are defined as `payable`. However, it is not possible to withdraw the ETH in the contracts .

Impact

This Issue could result in ETH being locked in the contract for an indefinite amount of time.

Preconditions

The Issue is likely if these payable functions are called with `Eth(msg.value > 0)`.

Mitigation

We recommend updating the functions to `nonpayable`.

Verification

Resolved.

Issue C: Updates Between Non-Zero Allowances Can Result in Exploits

Location

`contracts/token-implementations/ERC20.sol#L61`

Synopsis

The process by which a user updates from one non-zero allowance to another can be susceptible to exploits. When user A approves the transfer of N tokens to user B, and user A updates the allowance to M using the `approve` function, user B can deploy the `approve` transaction in the mempool and take M+N tokens by transferring N tokens just before the second approval with a higher gas price, and transferring M tokens after the second approval.

Impact

In this case, this Issue could result in user B stealing user A's M tokens.

Remediation

We recommend that the Hilter team prevent updates between non-zero allowances. The user planning to update the approval from one non-zero allowance to another must first set the allowance to `zero`. As a result, the user can detect if the allowance was used by the approved user before the new approval. For example:

```
   function approve(address spender, uint256 amount) external virtual
override returns (bool) {

        require(!((amount!= 0) && (allowance[msg.sender][_spender] != 0

        ))); _approve(msg.sender, spender, amount);

        return true;

    }
```

Verification

Resolved.

Suggestions

Suggestion 1: Add Check for Out-Of-Bound Values in getImplementation

Location

contracts/interchain-token-service/InterchainTokenService.sol#L210

Synopsis

In the `getImplementation` function, if the input value is beyond the range of `TokenManagerType` , `address(0)` is returned. As a result, when it is called in [TokenMangerProxy](), the `impl.delegatecall` function fails.

Mitigation

We recommend adding the `revert` function when the input value is not in the range of `TokenManagerType`.

Status

The Hilter team has added a condition, which reverts when the input value is beyond the range of `TokenManagerType`, as follows:

```
if (tokenManagerType > uint256(type(TokenManagerType).max)) revert
InvalidImplementation();
```

Verification

Resolved

Suggestion 2: Add Check To Validate the Parameters of addTrustedAddress

Location

`contracts/linker-router/LinkerRouter.sol#L77`

Synopsis

The function `addTrustedAddress` only checks the parameter length rather than the actual value of the chain and address, due to which `bytes32(0)` and `address(0)` can be passed. Consequently, this can result in the storing of `address(0)` for the mapping of `remoteAddresses[chain]`.

Mitigation

We recommend adding checks for the parameter of the function `addTrustedAddress`, such that `bytes32(0)` and `address(0)` can be prevented from being set for the chain and interchain token addressrespectively.

Verification

Resolved.

Suggestion 3: Resolve TODOs in Codebase

Synopsis

There are unresolved TODO items in the code comments of the in-scope implementation, which may lead to a lack of clarity and cause confusion about its completion. Resolving TODOs prior to a comprehensive security audit of the code allows security researchers to better understand the full intended functionality of the code, indicates completion, and increases readability and comprehension.

Mitigation

We recommend that TODOs be resolved or removed from the codebase.

Status

At the time of the verification, our team found that pending TODO's still persist in the InterTokenService smart contract.

Verification

Partially resolved.

Suggestion 4: Enable Two-Step Ownership Transfers

Location

`contracts/utils/Adminable.sol#L53`

Synopsis

The current method for changing admins only allows for a single transaction swap, which can lead to accidental changes to accounts that are not controlled by an admin.

Mitigation

We recommend allowing the option to propose and accept changes to important roles, before removing the old account, to ensure that the new account is controlled as expected.

Status

The Hilter team has created a custom implementation that allows for both single and two-step ownership transfers.

Verification

Resolved.

Suggestion 5: Update Implementation To Make It Consistent With the Specification Mentioned in the Comment

Location

`contracts/utils/ExpressCallHandler.sol#L15`

`contracts/interchain-token-service/InterchainTokenService.sol#L65`

Synopsis

The values of `PREFIX_EXPRESS_RECEIVE_TOKEN`, `PREFIX_EXPRESS_RECEIVE_TOKEN_WITH_DATA`, and `contractId` are not calculated as mentioned in the comment. These values are calculated after the subtraction of a constant value from the respective hashes.

Mitigation

We recommend calculating the values of `PREFIX_EXPRESS_RECEIVE_TOKEN`, `PREFIX_EXPRESS_RECEIVE_TOKEN_WITH_DATA`, and `ContractId` as mentioned in the comment to make the implementation consistent with the specification mentioned in the comment.

Status

The Hilter team has set the value of `ContractId` to - `keccak256('interchain-token-service')`, but the value of `PREFIX_EXPRESS_RECEIVE_TOKEN` is different from the value mentioned in comment, as illustrated below:

The value of the `PREFIX_EXPRESS_RECEIVE_TOKEN` is:
`0x67c7b41c1cb0375e36084c4ec399d005168e83425fa471b9224f6115af865612`

However, the value of `uint256(keccak256('prefix-express-give-token'))` is:
`0x67c7b41c1cb0375e36084c4ec399d005168e83425fa471b9224f6115af865612`

Verification

Partially Resolved.

Suggestion 6: Remove Unnecessary Modifier

Location

contracts/interchain-token-service/InterchainTokenService.sol#L411

Synopsis

The function `expressReceiveToken` does not have any trust assumptions since it is a helper function that forwards tokens from the relayer. Given that there is no attack surface for this functionality, there is no need to have the pause failsafe modifying it.

Mitigation

We recommend removing the unnecessary modifier.

Status

The Hilter team has removed the modifier as suggested.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.