SALUS SECURITY

DEC 2025

CODE SECURITY ASSESSMENT

NIZA GLOBAL

Overview

Project Summary

- Name: Nizaglobal - Bridge
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/nizaglobal/niza-bridge2.0
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Nizaglobal - Bridge
Version	v2
Type	Solidity
Dates	Dec 10 2025
Logs	Dec 09 2025; Dec 10 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	2
Total informational issues	2
Total	6

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Lack of rebalance mechanism	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Bridge overpays Hyperlane message fees without refunding excess fee	Low	Business Logic	Resolved
4	Third-party dependencies	Low	Dependency	Acknowledged
5	Missing events for functions that change critical state	Informational	Logging	Resolved
6	Mapping declaration lacks named parameters for improved readability	Informational	Code Quality	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Lack of rebalance mechanism	
Severity: Medium	Category: Business Logic
Target: - src/NizaVault.sol	

Description

According to the codebase, bridging tokens will increase the `totalLocked` and handling tokens will decrease the `totalAvailable`.

src/NizaVault.sol:L145-L193

```
function lockTokens(...) external onlyRouter whenNotPaused nonReentrant {
    …
    totalLocked += amount;
}
function releaseTokens(...) external onlyRouter whenNotPaused nonReentrant {
    …
    totalAvailable -= amount;
}
```

The NizaVault contract employs a strict double-entry bookkeeping system, artificially dividing the token balance within the contract into two parts:

1. `totalLocked`: Records funds locked when a user leaves the same blockchain.
2. `totalAvailable`: Records reserve funds injected by liquidity providers (LPs).

When processing cross-chain reverts (release tokens), the contract forcibly checks if totalAvailable is sufficient (`if (totalAvailable < amount) revert...`), completely ignoring the large amount of idle funds held in `totalLocked`. Due to the fungible nature of tokens, this isolation is logically unnecessary and could lead to situations where "the contract has funds, but users cannot withdraw them."

Proof of Concept:

```
function testHandleWithoutTotalAvailable() public {
    uint256 amount = 100 ether;

    vm.startPrank(user1);
    token.approve(address(router), amount);
    token.approve(address(vault), amount);

    bytes32 messageId = router.bridge{value: 0.001 ether}(
        amount,
        REMOTE_DOMAIN,
        user2
    );
    vm.stopPrank();
    INizaBridge.BridgeMessage memory message = INizaBridge.BridgeMessage({
        sender: user2,
        recipient: user1,
        amount: amount,
        nonce: 0,
        sourceChain: REMOTE_DOMAIN,
        timestamp: block.timestamp
    });

    bytes memory encodedMessage = abi.encode(message);
    bytes32 sender = bytes32(uint256(uint160(address(this))));

    vm.expectRevert(NizaVault.InsufficientVaultLiquidity.selector);
    mailbox.testDeliverMessage(
        REMOTE_DOMAIN,
        sender,
        address(router),
        encodedMessage
    );
}
```

Recommendation

Add a rebalance mechanism.

Status

The team has resolved this issue in commit 846b64d.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - src/NizaVault.sol	

Description

In the `NizaVault` contract, there exists one privileged role `Owner`. The owner has the authority to some key functions such as `emergencyWithdraw()`. If the role's private key is compromised, an attacker could trigger this function to drain all funds.

src/NizaVault.sol:L250-L261

```
function emergencyWithdraw(
   address token,
   uint256 amount,
   address recipient
) external onlyOwner whenPaused {
   if (recipient == address(0)) revert InvalidAmount();
   if (amount == 0) revert InvalidAmount();

   IERC20(token).safeTransfer(recipient, amount);

   emit EmergencyWithdrawal(token, amount, recipient);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team and states that will add RBAC instead of a multi sig in the future.

3. Bridge overpays Hyperlane message fees without refunding excess fee	
Severity: Low	Category: Business Logic
Target: - src/NizaBridgeRouter.sol	

Description

The `` `bridge()` `` function checks that `` `msg.value` `` is at least the required Hyperlane message fee returned by `` `mailbox.quoteDispatch()` ``, but then forwards the entire `` `msg.value` `` to `` `mailbox.dispatch()` ``.

src/NizaBridgeRouter.sol:L154-L213

```
function bridge(uint256 amount, uint32 destinationChain, address recipient)
   …
{
   …
   uint256 requiredFee = mailbox.quoteDispatch(
       destinationChain,
       encodedMessage
   );
   if (msg.value < requiredFee) revert InsufficientGasPayment();

   messageId = mailbox.dispatch{value: msg.value}(
       destinationChain,
       remoteRouter,
       encodedMessage
   );
   …
}
```

This means any excess fee sent by the user (e.g., due to a misconfigured frontend or manual overpayment) will not be refunded, users may consistently overpay bridging fees, leading to unnecessary loss of funds.

Recommendation

Use the quoted fee as the actual value sent to dispatch and return any excess fee to the user.

Status

The team has resolved this issue in commit [846b64d](846b64d).

4. Third-party dependencies	
Severity: Low	Category: Dependency
Target: - src/NizaBridgeRouter.sol	

Description

The `NizaBridgeRouter` contract relies on the `Hyperlane` protocol to enable the basic functionality. The current audit treats third-party entities as black boxes and assumes they are working correctly. However, in reality, third parties could be compromised, resulting in the disruption of token functionalities.

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to regularly monitor the statuses of third parties to reduce the impacts when they are not functioning properly.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Missing events for functions that change critical state	
Severity: Informational	Category: Logging
Target: - src/NizaBridgeRouter.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `` `NizaBridgeRouter` ``contract, the event is lacking in the privileged function `` `setSecurityModule()` ``.

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit 846b64d.

6. Mapping declaration lacks named parameters for improved readability	
Severity: Informational	Category: Code Quality
Target: - src/NizaBridgeRouter.sol	

Description

Since Solidity version 0.8.18, the language allows developers to add named key and value identifiers in mapping declarations using the syntax:

```
mapping(KeyType keyName => ValueType valueName)
```

This enhances code readability and clarifies how a mapping is intended to be used. In those contracts, the mappings are declared without named parameters, making the purpose of the key and value less obvious upon inspection.

src/NizaBridgeRouter.sol:L57-L61,L66-L67

```
/// @notice Mapping of supported chain IDs to their router addresses
mapping(uint32 => bytes32) public remoteRouters;
/// @notice Mapping of chain IDs to check if they're supported
mapping(uint32 => bool) public supportedChains;
/// @notice Processed message tracking to prevent replays
mapping(bytes32 => bool) public processedMessages;
```

Recommendation

Consider updating the mapping declaration to explicitly name the key and value identifiers, enhancing readability and making the mapping's purpose clearer.

Status

The team has resolved this issue in commit 846b64d.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 67ee6a3:

File	SHA-1 hash
BridgeConfig.sol	d063e66490a1e14c9ed2a86ca2c1857e6bb5b551
NizaVault.sol	ffd6ffc0c344b53ed86d1c4a80e350711161e9b1
NizaBridgeRouter.sol	19747f1d48ea120673a559e1c7024545b4f54ef8
TypeConverter.so	ebeefd8283acddccd47d88ef41aed29984e9b6e1