SALUS SECURITY

DEC 2025

CODE SECURITY ASSESSMENT

GATE

Overview

Project Summary

- Name: GateChain - SolConnector
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/gatechain/perps
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	GateChain - SolConnector
Version	v2
Type	Solidity
Dates	Dec 08 2025
Logs	Dec 01 2025; Dec 08 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	2
Total	4

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 issue

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Implementation contract lacks _disableInitializers()	Low	Business Logic	Resolved
2	Third-party dependencies	Low	Dependency	Acknowledged
3	Missing events for functions that change critical state	Informational	Logging	Acknowledged
4	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Implementation contract lacks _disableInitializers()

Severity: Low	Category: Configuration
Target: - sol-cc/SolConnector.sol	

Description

The `SolConnector` contract does not call OpenZeppelin's `_disableInitializers()` in its constructor.

As a result, anyone can send a direct transaction to the implementation address (not the proxy) and invoke `initialize()`, set `_lzEndpoint`, `_delegate` and `_solEid`. This is an extremely dangerous action.

Recommendation

Add a constructor that locks the implementation:

```
constructor() {
    _disableInitializers();
}
```

Status

The team has resolved this issue in commit 1431ea7.

2. Third-party dependencies

Severity: Low	Category: Dependency
Target: - sol-cc/SolConnector.sol	

Description

All contracts rely on the `OAppUpgradeable` contract 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. The team states that this is a trusted dependency.

2.3 Informational Findings

3. Missing events for functions that change critical state

Severity: Informational	Category: Logging
Target: - sol-cc/SolConnector.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 `SolConnector` contract, events are lacking in the privileged functions (e.g. `setLedger()`, `setOptions()` and `setSolEid()`).

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

4. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.22;
```

For example, the `SolConnector` uses a floating compiler version ^0.8.22.

Using a floating pragma ^0.8.22 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 5a478c2:

File	SHA-1 hash
sol-cc/library/MsgCodec.sol	136ef6a9c0618b85707a94fb12185453feb9f335
sol-cc/library/Utils.sol	158620d4382e9cde2afb984d9bfa54a797290208
sol-cc/SolConnector.sol	02d8bc6f03daeb2a4a0d27cdcee57b017192b518