SALUS SECURITY

DEC 2025

CODE SECURITY ASSESSMENT

GATE

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	GateChain - contract evm
Version	v2
Type	Solidity
Dates	Dec 08 2025
Logs	Dec 04 2025, Dec 08 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	4
Total informational issues	1
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	Centralization risk	Medium	Centralization	Acknowledged
2	Function `updateMarketUpload` cannot work as expected	Low	Business Logic	Acknowledged
3	Users may fail to deposit funds	Low	Business Logic	Acknowledged
4	Users may lose some native fess	Low	Business Logic	Acknowledged
5	Deposit limit may be not accurate	Low	Business Logic	Acknowledged
6	Redundant code	Informational	Redundancy	Acknowledged

2.2 Notable Findings

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

1. Centralization risk	
Severity: Medium	Category: Centralization
Target: - contract-evm/OperatorManager.sol - contract-evm/Ledger.sol	

Description

In the OperatorManager/Ledger contracts, there exists some privileged roles, `Owner`, `onlyOperator`, etc. These roles have the authority to execute some key functions such as `eventUpload`.

If these roles' private keys are compromised, an attacker could trigger these functions to withdraw all funds.

contract-evm/OperatorManager.sol:L167-L177

```
function eventUpload(
    EventTypes.EventUpload calldata data
) external override onlyOperator {
    _delegatecall(
        abi.encodeWithSelector(
            IOperatorManagerImplA.eventUpload.selector,
            data
        ),
        _getOperatorManagerStorage().operatorManagerImplA
    );
}
```

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.

2. Function `updateMarketUpload` cannot work as expected

Severity: Low	Category: Business Logic
Target: - contract-evm/MarketManager.sol	

Description

In MarketManager, function `updateMarketUpload` is used to record unitary fundings. The function is expected to be called by OperatorManager.

The problem here is that contract `OperatorManager` misses one interface to trigger `updateMarketUpload`. This will cause that function `updateMarketUpload` cannot be used as expected.

Similar issues happen in the function `setLastFundingUpdated`.

contract-evm/MarketManager.sol:L43-L52

```
function updateMarketUpload(MarketTypes.UploadSumUnitaryFundings calldata data) external
onlyOperatorManager {
    uint256 length = data.sumUnitaryFundings.length;
    for (uint256 i = 0; i < length; i++) {
        MarketTypes.SumUnitaryFunding calldata sumUnitaryFunding =
data.sumUnitaryFundings[i];
        MarketTypes.PerpMarketCfg storage cfg =
perpMarketCfg[sumUnitaryFunding.symbolHash];
        cfg.setSumUnitaryFundings(sumUnitaryFunding.sumUnitaryFunding);
        cfg.setLastMarkPriceUpdated(sumUnitaryFunding.timestamp);
    }
    emit FundingData(data.maxTimestamp);
}
```

Recommendation

Add one interface to trigger these functions.

Status

The team stated that these functions will not be used.

3. Users may fail to deposit funds	
Severity: Low	Category: Business Logic
Target: - contract-evm/vaultSide/Vault.sol	

Description

In the Vault contract, users can deposit assets via the function `deposit`. The owner can enable or disable deposit fees. If the deposit fee is disabled, the crossChainRelay will pay the cross chain fees via native tokens.

The problem here is that malicious users can keep depositing tiny amounts of tokens to consume all native fees in crossChainRelay. After that, when other users want to deposit funds, their deposit operation will be reverted because cross chain relay does not have enough native fees to pay.

contract-evm/MarketManager.sol:L43-L52

```
function _deposit(
    address receiver,
    VaultTypes.VaultDepositFE calldata data
) internal whenNotPaused {
    if (depositFeeEnabled) {
        ...
    } else {
        // Call regular deposit function
        IVaultCrossChainManager(crossChainManagerAddress).deposit(
            depositData
        );
    }
}
```

Recommendation

Keep enabling deposit fees.

Status

This issue has been acknowledged by the team.

4. Users may lose some native fees	
Severity: Low	Category: Business Logic
Target: - contract-evm/vaultSide/Vault.sol	

Description

In the Vault contract, users can deposit funds via the function `deposit`. When the deposit fee is enabled, users can transfer some native ether as the cross chain fees.

The problem here is that if users deposit with some native fees after the owner disables deposit fees, these native fees will be left in this contract.

contract-evm/OperatorManager.sol:L167-L177

```
function _deposit(
    address receiver,
    VaultTypes.VaultDepositFE calldata data
) internal whenNotPaused {
    if (depositFeeEnabled) {
        // Revert if no fees provided
        if (msg.value == 0) revert ZeroDepositFee();
        IVaultCrossChainManager(crossChainManagerAddress)
            .depositWithFeeRefund{value: msg.value}(
            msg.sender, // refund receiver.
            depositData
        );
    } else {
        // Call regular deposit function
        IVaultCrossChainManager(crossChainManagerAddress).deposit(
            depositData
        );
    }
}
```

Recommendation

Add input check in function `deposit`. If the deposit fee is disabled, `msg.value` should be 0.

Status

This issue has been acknowledged by the team.

5. Deposit limit may be not accurate	
Severity: Low	Category: Business Logic
Target: - contract-evm/vaultSide/Vault.sol	

Description

In the Vault contract, users can deposit funds via the function `deposit`. If the total deposit amount exceeds the deposit limit, users' deposit operation will be reverted.

The problem here is that when users withdraw assets, some assets will be left in the vault contract as the withdrawal fees. It means that even if nobody deposits here, there may be some left tokens here. This will cause the deposit limit's calculation to be inaccurate.

contract-evm/OperatorManager.sol:L327-L334

```
function _deposit(
    address receiver,
    VaultTypes.VaultDepositFE calldata data
) internal whenNotPaused {
    if (
        tokenAddress2DepositLimit[address(tokenAddress)] != 0 &&
        data.tokenAmount + tokenAddress.balanceOf(address(this)) >
        tokenAddress2DepositLimit[address(tokenAddress)]
    ) {
        // Revert if deposit limit exceeded
        revert DepositExceedLimit();
    }
}
```

Recommendation

Record the total withdrawal fees and exclude withdrawal fees when calculating the deposit limit.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

6. Redundant code	
Severity: Informational	Category: Redundancy
Target: - contract-evm/vaultSide/Vault.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following functions are not being utilized:

contract-evm/vaultSide/Vault.sol:L313-L317

```
function deposit(
       VaultTypes.VaultDepositFE calldata data
   ) public payable override whenNotPaused {
}
function _deposit(
    address receiver,
    VaultTypes.VaultDepositFE calldata data
   ) internal whenNotPaused {
}
```

contract-evm/vaultSide/Vault.sol:L313-L317

```
contract Vault is
   IVault,
   PausableUpgradeable,
   OwnableUpgradeable,
   ReentrancyGuardRevised,
   AccessControlRevised,
   Version,
   UUPSUpgradeable
{
}
```

Recommendation

Consider removing the redundant code.

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
FeeManager.sol	7cc2d797a8ad84cc02644c30a307380d476f3de5
Ledger.sol	2ae63e95acf07a76c4cf588be3bab5c27281268f
LedgerComponent.sol	e117b5ecb4f075cfa361f29c549e88bd12d5a4e8
LedgerImplA.sol	07d23b7827622f651dec76b049acd31be035dda9
LedgerImplB.sol	af265c31857b600e9968f4f156eec7a6c8d69a41
LedgerImplC.sol	1485de3fb6d2c8c5447192a4389bb15ca22b0c75
MarketManager.sol	e0c825d7bacf9e79fe13f088769b40a55d43d42d
OperatorManager.sol	dfb80da6638b8fb2df014cff46078ffebae77264
OperatorManagerImplA.sol	fab3d3244058226e8532fd3c8f02b7a2142ad4f3
OperatorManagerImplB.sol	e69bff2b42fcde95bba40e0c7cdf3db8dd2e3e99
VaultManager.sol	713d83134c2630a68b585a5a3806344680e073f3
Vault.sol	718b000cd91d3c74487027ac663d0665c8a4f891