SALUS SECURITY

JAN 2025

CODE SECURITY ASSESSMENT

INCEPTIONLRT

Overview

Project Summary

- Name: InceptionLRT- Incremental Audit
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/inceptionlrt/smart-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	InceptionLRT- Incremental Audit
Version	v3
Type	Solidity
Dates	Jan 14 2025
Logs	Jan 02 2025; Jan 10 2025; Jan 14 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	3
Total informational issues	3
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	Low	Centralization	Acknowledged
2	Missing check when configuring ratios	Low	Business Logic	Resolved
3	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
4	Error Message and Functionality Mismatch	Informational	Inconsistency	Resolved
5	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved
6	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

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

1. Centralization risk	
Severity: Low	Category: Centralization
Target: - projects/bridge-lz/contracts/FerryAdapter/FraxFerryLZCrossChainAdapterL2.sol - projects/vaults/contracts/rebalancer/ERC20RebalancerStorage.sol - projects/vaults/contracts/assets-handler/InceptionOmniAssetsHandler.sol	

Description

The `` `FraxFerryLZCrossChainAdapterL2` ``, `` `ERC20RebalancerStorage` `` and `` `InceptionOmniAssetsHandler` `` contracts contain privileged addresses. The owner has the authority to modify all critical parameters within the contract, such as setting the default adapter, changing the inception token, altering the ferry, and more.

If the private key of either admin or owner is compromised, an attacker could manipulate these critical parameters, disrupt the normal operation of the contract, and harm the interests of regular users.

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. Missing check when configuring ratios

Severity: Low	Category: Business Logic
Target: - projects/vaults/contracts/vaults/InceptionERC20OmniVault.sol	

Description

projects/vaults/contracts/vaults/InceptionERC20OmniVault.sol:L429 - L473

```
function setDepositBonusParams(
    uint64 newMaxBonusRate,
    uint64 newOptimalBonusRate,
    uint64 newDepositUtilizationKink
) external onlyOwner {
    if (newMaxBonusRate > MAX_PERCENT)
        revert ParameterExceedsLimits(newMaxBonusRate);
    if (newOptimalBonusRate > MAX_PERCENT)
        revert ParameterExceedsLimits(newOptimalBonusRate);
    if (newDepositUtilizationKink > MAX_PERCENT)
        revert ParameterExceedsLimits(newDepositUtilizationKink);

    maxBonusRate = newMaxBonusRate;
    optimalBonusRate = newOptimalBonusRate;
    depositUtilizationKink = newDepositUtilizationKink;

    emit DepositBonusParamsChanged(
        newMaxBonusRate,
        newOptimalBonusRate,
        newDepositUtilizationKink
    );
}

function setFlashWithdrawFeeParams(
    uint64 newMaxFlashFeeRate,
    uint64 newOptimalWithdrawalRate,
    uint64 newWithdrawUtilizationKink
) external onlyOwner {
    if (newMaxFlashFeeRate > MAX_PERCENT)
        revert ParameterExceedsLimits(newMaxFlashFeeRate);
    if (newOptimalWithdrawalRate > MAX_PERCENT)
        revert ParameterExceedsLimits(newOptimalWithdrawalRate);
    if (newWithdrawUtilizationKink > MAX_PERCENT)
        revert ParameterExceedsLimits(newWithdrawUtilizationKink);

    maxFlashFeeRate = newMaxFlashFeeRate;
    optimalWithdrawalRate = newOptimalWithdrawalRate;
    withdrawUtilizationKink = newWithdrawUtilizationKink;

    emit WithdrawFeeParamsChanged(
        newMaxFlashFeeRate,
        newOptimalWithdrawalRate,
        newWithdrawUtilizationKink
    );
}
```

The `InceptionERC20OmniVault` contract lacks a check when configuring the ratio regarding deposit rewards and withdrawal fees, if the ratio of rewards on deposits is greater than the ratio of fees on withdrawals, a malicious user may cause all accumulated rewards in the contract to be emptied through constant deposits and withdrawals.

Recommendation

It is recommended to add a check for proportionality in the configuration function.

Status

The team has resolved this issue in commit f6506e2.

3. Implementation contract could be initialized by everyone	
Severity: Low	Category: Business Logic
Target: - projects/bridge-lz/contracts/FerryAdapter/FraxFerryLZCrossChainAdapterL2.sol - projects/vaults/contracts/rebalancer/ERC20Rebalancer.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the `` `initialize()` `` function in `` `ERC20Rebalancer` ``, `` `FraxFerryLZCrossChainAdapterL2` `` 's implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the `` `_disableInitializers()` `` function in the constructor of the contract to automatically lock it when it is deployed.

Status

The team has resolved this issue in commit b1a2c57.

2.3 Informational Findings

4. Error Message and Functionality Mismatch	
Severity: Informational	Category: Inconsistency
Target: - projects/vaults/contracts/rebalancer/ERC20Rebalancer.sol	

Description

The `updateTreasuryData()` function validates transaction timestamps, but its checks don't match the error message—different validations all return `MissingOneOrMoreL2Transactions`.

projects/vaults/contracts/rebalancer/ERC20Rebalancer.sol:L52 - L59

```
require(
    txData.timestamp != 0,
    MissingOneOrMoreL2Transactions(defaultChainId)
);
require(
    block.timestamp - txData.timestamp <= assetInfoTxMaxDelay,
    MissingOneOrMoreL2Transactions(defaultChainId)
);
```

Recommendation

Consider adjusting the error messages for different validations.

Status

The team has resolved this issue in commit fc49179.

5. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - projects/vaults/contracts/vaults/InceptionERC20OmniVault.sol	

Description

The `` `InceptionERC20OmniAssetsHandler` `` contract inherits from the `` `OwnableUpgradeable` `` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit 6fa9e26.

6. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - projects/vaults/contracts/rebalancer/ERC20Rebalancer.sol - projects/vaults/contracts/rebalancer/ERC20RebalancerStorage.sol - projects/vaults/contracts/vaults/InceptionERC20OmniVault.sol	

Description

```
pragma solidity ^0.8.27;
pragma solidity ^0.8.20;
```

`ERC20Rebalancer` and `ERC20RebalancerStorage` contracts use a floating compiler version `^0.8.27`. The `InceptionERC20OmniVault` contract uses a floating compiler version `^0.8.20`.

Using a floating pragma `^0.8.27` and `^0.8.20` 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

The team has resolved this issue in commit 44f8d17.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 08b1f20:

File	SHA-1 hash
projects/bridge-lz/contracts/abstract/AbstractFraxFerryERC20Adapter.sol	d2ccd9ebf5e1db5f385fd02aafaa331131a30245
projects/bridge-lz/contracts/FerryAdapter/FraxFerryLZCrossChainAdapterL2.sol	f6f67a03b2fc3c55b22636cd0d72b397316b5982
projects/vaults/contracts/rebalancer/ERC20Rebalancer.sol	bc6f03b6540f0cb74ae134816cdae44c56880f86
projects/vaults/contracts/rebalancer/ERC20RebalancerStorage.sol	bf1ea826aef532f3ade6a58b5a06036c0afe1dfb
projects/vaults/contracts/vaults/InceptionERC20OmniVault.sol	80abf153629a637d947ae1dec75af693d35b4313