SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

AVALON FINANCE

Overview

Project Summary

- Name: Avalon Finance - USDA Pool
- Platform: Merlin chain
- Language: Solidity
- Repository:
  - https://github.com/avalonfinancexyz/USDAPool
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Avalon Finance - USDA Pool
Version	v3
Type	Solidity
Dates	Oct 09 2024
Logs	Sep 30 2024; Oct 08 2024; Oct 09 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	4
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	Precision loss	Low	Numerics	Resolved
2	Repeated interest calculation	Low	Business Logic	Resolved
3	Prevent USDT providers from earning profits	Low	Business Logic	Acknowledged
4	Centralization risk	Low	Centralization	Acknowledged
5	Missing events for functions that change critical state	Informational	Logging	Acknowledged
6	Events are not indexed	Informational	Logging	Acknowledged

2.2 Notable Findings

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

1. Precision loss	
Severity: Low	Category: Numerics
Target: - contracts/LendingPool.sol	

Description

The contract has multiple occurrences of precision loss issues.
The functions `redeemUSDT` and `redeemUSDTFrom` have precision loss issues when calculating share, due to the downward rounding in division.
Due to downward rounding, it is possible to burn 0 shares when redeeming a dust amount of USDT.

contracts/LendingPool.sol:L131-L145

```
function redeemUSDT(
    uint256 _amount
) external onlyRole(LENDER_ROLE) realizeInterests {
    uint256 redeemShares = (_amount * totalDepositedUSDTShares) /
        totalDepositedUSDT;
    require(
        usdtDepositedShares[msg.sender] >= redeemShares,
        "redeem amount exceeds balance"
    );
    usdtDepositedShares[msg.sender] -= redeemShares;
    …
}
```

contracts/LendingPool.sol:L151-L66

```
function redeemUSDTFrom(
    address user,
    uint256 _amount
) external onlyRole(ADMIN_ROLE) realizeInterests {
    uint256 redeemShares = (_amount * totalDepositedUSDTShares) /
        totalDepositedUSDT;
    require(
        usdtDepositedShares[user] >= redeemShares,
        "redeem amount exceeds balance"
    );
    usdtDepositedShares[user] -= redeemShares;
    …
}
```

In the `realizeInterests`, `getSharesByAmount`, and `getAmountByShare` functions, when calculating the additional `totalInterest`, precision is lost due to dividing before multiplying. By multiplying first and then dividing, higher precision can be maintained.

contracts/LendingPool.sol:L230-L232; contracts/LendingPool.sol:L249-L251;

contracts/LendingPool.sol:L267-L269

```
uint256 totalInterest = ((apr * totalBorrowedUSDT) /
    365 days /
    APR_COEFFICIENT) * (block.timestamp - lastCheckpoint);
```

Recommendation

We recommend changing the rounding direction for share calculations and altering the calculation of `` `totalInterest` `` from dividing before multiplying to multiplying before dividing. However, modifying the rounding direction requires adding an initialization mint to prevent manipulation.

Status

This issue has been resolved by the team with commit 5c0042a.

2. Repeated interest calculation	
Severity: Low	Category: Business Logic
Target: - contracts/LendingPool.sol	

Description

The main function of the modifier `realizeInterests()` is to calculate and update the interest in the lending pool, including the `totalBorrowedUSDT` amount, `totalDepositedUSDT`, `unpaidInterestUSDT`. The `totalBorrowedUSDT` should represent the actual total amount of USDT loaned out. Interest should be a separate concept and should not be directly added to the `totalBorrowedUSDT`. This will lead to double calculation of interest, causing the project team to pay more interest than originally intended.

contracts/LendingPool.sol:L228-L239

```
modifier realizeInterests() {
    if (totalBorrowedUSDT != 0) {
        uint256 totalInterest = ((apr * totalBorrowedUSDT) /
            365 days /
            APR_COEFFICIENT) * (block.timestamp - lastCheckpoint);
        totalBorrowedUSDT += totalInterest;
        totalDepositedUSDT += totalInterest;
        unpaidInterestUSDT += totalInterest;
    }
    lastCheckpoint = block.timestamp;
    _;
}
```

Recommendation

Remove the `totalBorrowedUSDT` update.

Status

This issue has been resolved by the team with commit 2d94d64.

3. Prevent USDT providers from earning profits	
Severity: Low	Category: Business Logic
Target: - contracts/LendingPool.sol	

Description

The `redeemUSDA` function allows the borrower to redeem an equivalent amount of USDA tokens by transferring in USDT, while simultaneously updating the `totalBorrowedUSDT` and limit records.

contracts/LendingPool.sol:L172-L186

```
function redeemUSDA(
     uint256 _amount
) external onlyRole(BORROWER_ROLE) realizeInterests {
    IERC20(USDT).safeTransferFrom(msg.sender, address(this), _amount);
    if (totalBorrowedUSDT >= _amount) {
        totalBorrowedUSDT -= _amount;
    } else {
        additionalUSDT += (_amount - totalBorrowedUSDT);
        totalBorrowedUSDT = 0;
    }
    …
}
```

The interest is related to the `totalBorrowedUSDT`.

contracts/LendingPool.sol:L228-L239

```
modifier realizeInterests() {
    if (totalBorrowedUSDT != 0) {
        uint256 totalInterest = ((apr * totalBorrowedUSDT) /
            365 days /
            APR_COEFFICIENT) * (block.timestamp - lastCheckpoint);
        totalBorrowedUSDT += totalInterest;
        totalDepositedUSDT += totalInterest;
        unpaidInterestUSDT += totalInterest;
    }
    lastCheckpoint = block.timestamp;
    _;
}
```

Attach Scenario

Alice deposits 1000 USDT, and both Bob and Carol each borrow 500 USDT. At this point, interest calculation begins. but Bob is able to repay more than his original loan—1000 USDT. As a result, the system's `totalBorrowedUSDT` will be reset to zero, and interest calculation ends. However, Carol's actual loan still exists, but it will no longer accrue interest. This would cause Alice to lose interest.

Recommendation

Over-repayment is not allowed.

Status

This issue has been acknowledged by the team.

4. Centralization risk	
Severity: Low	Category: Centralization
Target: - contracts/LendingPool.sol	

Description

There is a privileged owner role in the `LendingPool` contract. The owner of the `LendingPool` contract can reduce the `apr` to zero. This would cause users to lose interest earnings.

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.

2.3 Informational Findings

5. Missing events for functions that change critical state	
Severity: Informational	Category: Logging
Target: - contracts/LendingPool.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 `LendingPool` contract, events are lacking in the privileged setter functions (e.g. `distributeInterests` function, and `realizeInterests` modifier).

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

6. Events are not indexed	
Severity: Informational	Category: Logging
Target: - contracts/LendingPool.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. The emitted events are not indexed, making off-chain scripts such as front-ends of dApps difficult to filter the events efficiently.

In the `LendingPool` contract, the `RequestUSDT` , `ClaimUSDT`, `DepositUSDT`, `RedeemUSDT` and `RedeemUSDA` events are not indexed.

Recommendation

Consider adding the indexed keyword in the `RequestUSDT` , `ClaimUSDT`, `DepositUSDT`, `RedeemUSDT` and `RedeemUSDA` events.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 1feda66:

File	SHA-1 hash
contracts/LendingPool.sol	927650f8f5c81234340dd066af70c0bbadb8d6c0
contracts/token/ERC20Token.sol	e63d95ecb9e7921260398d7fe4ca7714eaf38808