SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

AVALOAN FINANCE

Overview

Project Summary

- Name: Avaloan Finance - AALoan
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/avalonfinancexyz/AALoan
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Avaloan Finance - AALoan
Version	v2
Type	Solidity
Dates	Oct 17 2024
Logs	Oct 14 2024; Oct 17 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	1
Total informational issues	2
Total	5

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	The getPoolManagerReserveInformation function may be susceptible to a dos attack	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	The setEmergencyController function did not correctly transfer the authority	Low	Access Control	Acknowledged
4	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledged
5	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. The getPoolManagerReserveInformation function may be susceptible to a dos attack	
Severity: Medium	Category: Business Logic
Target: - src/protocol/PoolManager.sol	

Description

src/protocol/PoolManager.sol:L527-L524

```
function getPoolManagerReserveInformation()
    external
    view
    returns (
        DataTypes.PoolManagerReserveInformation
            memory poolManagerReserveInfor
    )
{
    uint256 length = userList.length;
    for (uint i = 0; i < userList.length; i++) {
        poolManagerReserveInfor.debt += getUserPoolReserveInformation(
            userList[i]
        ).debt;
    }
    poolManagerReserveInfor.userAmount = userAmount;
    poolManagerReserveInfor.collateral = totalCollateral;
    poolManagerReserveInfor.claimableBTC = totalClaimableBTC;
}
```

The `getPoolManagerReserveInformation()` function calculates debt by looping through `userList`, including interest calculated through the internally called `getUserPoolReserveInformation` function, which increases the number of iterations due to compounding. If the `userList` length is large, this may cause the function to become unusable, and there are no functions in the contract that can reduce the `userList.length`.

Recommendation

It is recommended to implement batching for the `userList`. When the `userList` is too long, the function can continue to operate by processing the list in segments.

Status

The team has resolved this issue in commit 0a814ea.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - src/protocol/FBTCOracle.sol - src/protocol/PoolManager.sol - src/protocol/PoolManagerConfigurator.sol	

Description

These contracts present a centralization risk as there are privileged addresses with the authority to modify critical variables in the contracts. Both `PoolManagerConfigurator` and `PoolManager` contracts utilize OpenZeppelin's AccessControl module, where the defaultAdmin has the ability to grant any permissions to any address.

If the `owner` or `defaultAdmin`'s private key is compromised, an attacker could modify the expiration time of the oracle information, change critical variables in the contract, or directly wipe out users' `collateral` and `debt`, among other malicious actions.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other 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.

3. The setemergencycontroller function did not correctly transfer the authority	
Severity: Low	Category: Access Control
Target: - src/protocol/PoolManagerConfigurator.sol	

Description

src/protocol/PoolManagerConfigurator.sol:L149 - L154

```
function setEmergencyController(
    address newEmergencyController
) external onlyRole(ADMIN_ROLE) {
    emergencyController = newEmergencyController;
    emit EmergencyControllerSet(newEmergencyController);
}
```

The `setEmergencyController()` function did not grant the new address the necessary role, which will prevent the new `emergencyController` from successfully calling the `pause()` and `unpause()` functions. This may cause potential losses to the project in certain emergency situations.

Although `defaultAdmin` also has permission to call this function, it could lead to confusion in permission assignments.

Recommendation

It is recommended that the `EMERGENCY_CONTROLLER_ROLE` permission be granted for `newEmergencyController` in the `setEmergencyController()` function. Alternatively, consider using the `onlyEmergencyController` modifier.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

4. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - src/protocol/FBTCOracle.sol	

Description

The `FBTCOracle` contract inherits from the `Ownable` 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 Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

5. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.13;
```

All contract use a floating compiler version `^0.8.13`.

Using a floating pragma `^0.8.13` 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 13f633c:

File	SHA-1 hash
src/protocol/FBTCOracle.sol	b8401bb6d4778cbdf00591c13620f2af18c751d4
src/protocol/PoolManager.sol	13c6560caa5f97734818abad4c607ea0ea0e2f88
src/protocol/PoolManagerConfigurator.sol	02ee77f16f0c1aca39059660ee91c59f18f0f098
src/protocol/PoolManagerStorage.sol	e08ec0fdc23f989bfa644f0f62e2321ecb4f2c66
src/protocol/library/type/DataTypes.sol	494f35891e6362d66ed0254073da70b77f661518