Fusion Finance Review

Review Resources:
A github repo containing protocol documentation, smart contracts and web app was provided.
Project Author:

- John Nyugen (jooohneth)

Project Auditor:

- Ahiara Ikechukwu Marvellous

Table of Contents

Fusion Finance Review
Table of Contents
Review Summary
Scope
Code Evaluation Matrix
Findings Explanation
Critical Findings
High Findings
Medium Findings
Low Findings
Gas Savings Findings
Informational Findings
Final remarks
Appendix and FAQ

Review Summary

Fusion Finance

A decentralised lending and borrowing protocol for users to lend and borrow Base Asset. A completely trustless environment for both lenders and borrowers, built on top of Ethereum and Ethereum L2 solutions. Lenders supply the protocol with Base Asset and in return earn interest in a form of $FUSN tokens, protocol's governance token.

The main branch of the fusion finance repo was reviewed 2 days ago. All main contracts, FusionCore.sol and FusionToken.sol were covered.

Scope

Code repo
Commit

The commit reviewed was 12cfb4d996650fb2abe0732eef78de54df6ba2ba. The review covered the repository at the specific commit and focused on the contracts directory.
All findings were presented to the john.

The review is a code review of smart contracts to identify potential vulnerabilities in the code.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Okay	Access control was applied only on one contract. Ownable is used to limit mint functionality in FusionToken.sol
Mathematics	Good	Solidity 0.8.9 is used, which provides overflow and underflow protection. Lending and Borrowing maths checks out. No low-level bitwise operation was used.
Libraries	Good	OpenZeppelin contracts such as ERC20, Ownable, IERC20 are imported. Chainlink's AggregatorV3Interface was used for ETH/USD price feed
Complexity	Low	Asides the mathematical calculations and Chainlink's integration no extra complexity was added. No proxy contracts pr delegatecalls used.
Documentation	Good	Comments existed for most functions and state variables and clarified what they did.
Monitoring	Good	Events exist for all important functions that modify state variables.
Testing	Good	All tests were passing and test coverage was expansive.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas Savings
    - Findings that can improve the gas efficiency of the contracts
- Informational
    - Findings including recommendations and best practices

Gas Savings

1. **withdrawLend function in FusionCore.sol**

   Proof of concept
   Unnecessary initialization of local variable to store _amount argument.

```
function withdrawLend(uint256 _amount) public {
        require(isLending[msg.sender], "Can't withdraw before lending!");
        require(
            lendingBalance[msg.sender] >= _amount,
            "Insufficient lending balance!"
        );

        uint256 yield = calculateYieldTotal(msg.sender);
        fusionBalance[msg.sender] += yield;
        startTime[msg.sender] = block.timestamp;

        lendingBalance[msg.sender] -= _amount;

        if (lendingBalance[msg.sender] == 0) {
            isLending[msg.sender] = false;
        }

        require(baseAsset.transfer(msg.sender, _amount), "Transaction
failed!");

        emit WithdrawLend(msg.sender, _amount);
    }
```

   Impact
   Gas savings, 100 gas saved

Informational Findings

1. Solc version 0.8.9 not recommended for deployment.

   Recommendation
   Use solc version 0.8.13-0.8.16 for bug fixes, compiler optimization and contract bytecode optimizations

2. **Add Comments to the calculateLiquidationPoint function.**

   Clarification needed on what the function does.

3. **State change via events after external call**

   Impact
   None but recommended practice.

   Recommendation
   Emit events before external calls.

Final Remarks

Having reviewed the contracts, no critical vulnerabilities were found across the main contracts with openzeppelin and chainlink integrations. The maths behind lending and borrowing work as expected. Test covers all functions and edge cases that can occur with chainlink.