QuillAudits

Audit Report
December, 2024

For

SIRIO

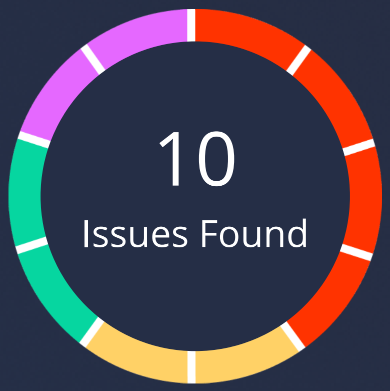
Table of Content

Table of Content

Executive Summary

Project Name	Sirio
Overview	Sirio protocol is the compound-a-like protocol on the Hedera blockchain.
Timeline	14th October 2024 - 4th Novemeber 2024
Updated Code Received	6th December 2024
Second Review	7th December 2024 - 9th December 2024
Audit Scope	The scope of this audit was to analyse Sirio Finance smart contract’s codebase for quality, security, and correctness.
Source Code	https://github.com/SirioFinance/Back-End
Commit Hash	0817aebaf7f53220dcd04228f93c686d58bb629b
Fixed In	https://github.com/SirioFinance/Back-End/tree/staging

Number of Security Issues per Severity



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	**4**	**2**	**2**	**2**

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Use of tx.origin
- ✓ Exception disorder
- ✓ Gasless send
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Using inline assembly
- ✓ ERC20 transfer() does not return boolean
- ✓ ERC20 approve() race
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw

Techniques and Methods

Throughout the audit of Sirio Finance smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Mythril, Slither, SmartCheck, Surya, Solhint.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Malicious user can ddos borrow and liquidation of every market of the sirio finance.

Path

contracts/MarketPositionManager.sol#L623, contracts/MarketPositionManager.sol#L681

Path

validateLiquidate, validateBorrow

Description

The validateLiquidate and validateBorrow functions are called by the liquidate and borrow functions, respectively, to determine whether liquidation or borrowing is feasible based on the current state of the smart contract. If these functions return false, liquidation and borrowing are not allowed.

However, the validation process is vulnerable to a denial-of-service (DoS) attack. Both functions check if the borrower is part of the borrowerList array, which is a dynamically growing list. A malicious user can exploit this by supplying funds to various markets from multiple addresses and borrowing small amounts from each. This action could repeatedly expand the borrowerList array, making the validateBorrow function increasingly gas-intensive. Eventually, this could prevent legitimate users from borrowing or liquidating, effectively halting these functionalities.

Recommendation

We recommend to make borrowerList as the mapping or use EnumerableSet.AddressSet to make it efficient.

Status

Resolved

2. Precision loss leads to dust in the protocol after liquidation.

Path

contracts/MarketPositionManager.sol#L445

Path

liquidateBorrow

Description

In liquidateBorrow function, liquidateAmountUSD has been calculated in such a manner that leads to precision loss because of that dust remain in the smart contract. A POC to verify this has been implemented **here**. Due to this issue after every liquidation dust will gather and with the time it could affects the exchange rate and interest earn by the users.

Recommendation

We recommend to re-writing the way of calculating the liquidateAmountUSD as below:

```
        uint256 M = MATH_SCALING_FACTOR;
        uint256 blr = borrowerLiquidationRisk;
        uint256 b = borrowedBalance;
        uint256 t = tokenPrice;

        liquidateAmountUSD = t * B * ( (M - blr) * M   +  M)
                                       -------------
                                            blr
                             ----------------------------------
                                  M * 10 ** underlyingDecimals

After simplifying the above formula it derives the below formula which
sorts out precision loss issues.

uint256 liquidateAmountUSD = b * M * tokenPrice / blr * 10 **
underlyingDecimals;
```

Status

Resolved

3. Precision loss during the calculation of shareAmount.

Path

contracts/BaseProtocol.sol#L797

Path

_calculateAmountInShares

Description

In the current implementation the shareAmount gets calculated using the token's decimal precision which has the possibility of less than 18 decimal. Because of this it leads to incorrect calculation of the share amount for redemption and causes dust remain in the contract and in some cases it also revert the parent function call.

Recommendation

scale the value with at least 18 decimals to avoid the precision loss as most of the places the precision is depend on the token decimal which will be less than 18 in many scenario and leads to loss of precision. We also highly recommend to scale mathematics to avoid the precision issues and it will be not be dependent on the token decimals.
Similar maths has been used in the compound and we recommend to follow the similar approach.
Reference: https://github.com/compound-finance/compound-money-market/blob/241541a62d0611118fb4e7eb324ac0f84bb58c48/contracts/Exponential.sol#L18

Status

Resolved

4. Protocol is reading stale prices leads to disaster for protocol.

Path

contracts/MarketPositionManager.sol#L1008
contracts/MarketPositionManager.sol#L1020

Path

_getUSDPrice

Description

The price is fetched from the Supra oracle, which returns a priceFeed data type containing a time field to indicate the last update timestamp. However, the protocol lacks a defined threshold for staleness—a time limit within which it considers price data as acceptable. Without this threshold, the protocol doesn't verify the time value, allowing it to use outdated prices, even if they are months old, as long as they are non-zero. This could lead to a scenario where the protocol accumulates significant bad debt, making recovery impossible.

Recommendation

We recommend to have a timestamp check and have a notion of threshold for staleness to know the acceptable old prices time limit. We also recommend to move fallback price check in the oracle contract itself to make MarketPositionManager contract more readable.

Status

Resolved

Medium Severity Issues

5. Supplier can supply more funds than designated maxProtocolSupplycap

Path

contracts/HBARProtocol.sol#L96 , contracts/SFProtocolToken.sol#L100

Function

supplyUnderlyingNative, supplyUnderlying

Description

In the supplyUnderlyingNative and supplyUnderlying functions, a check is performed to ensure that the provided _underlyingAmount does not cause the market's underlying balance to exceed the maxProtocolSupplyCap. However, in the current implementation of both functions, it is possible to supply an amount of the underlying token that surpasses the maxProtocolSupplyCap. As a result, this critical upper bound check may be bypassed, causing the protocol to operate outside its intended specifications.

Recommendation

We recommend to have greaterThanEqualTo i.e >= operand in the if statement at line contracts/HBARProtocol.sol#L96 & contracts/SFProtocolToken.sol#L100 instead of >.

Status

Resolved

6. Centralization risk leads to liquidation of all positions

Path

contracts/MarketPositionManager.sol#L192, contracts/MarketPositionManager.sol#L209

Function

setLoanToValue, setLiquidationIncentive

Description

The setLoanToValue and setLiquidationIncentive functions are restricted to be callable only by the owner, who has the freedom to assign any value to the loan-to-value (LTV) ratio for a given token. This includes setting the LTV to 1, which would render all positions using that token as collateral vulnerable to liquidation. The protocol could then perform these liquidations autonomously through bots, earning substantial liquidation incentives. Additionally, in conjunction with setLiquidationIncentive, the owner gains even greater control, allowing them to apply a maximum discount—any value less than 100—thereby amplifying their advantage in liquidation events.

Recommendation

We recommend to have some governance involved in changing these crucial values..

Status

Resolved

Low Severity Issues

7. Missing re-entrancy checks

Path

contracts/HBARProtocol.sol#L137, contracts/HBARProtocol.sol#L145,
contracts/HBARProtocol.sol#L203,
contracts/HBARProtocol.sol#L211,
contracts/SFProtocolToken.sol#L142,
contracts/SFProtocolToken.sol#L150

Function

redeem, redeemExactUnderlying, repayBorrowNative, repayBorrowBehalfNative

Description

All the functions described above are missing re-entrancy guard as it is necessary to have because in all these functions funds are moved to unknown third party address which can be malicious so to avoid re-entrancy issues it is required to have re-entrancy guard for the functions described above.

Recommendation

We recommend to add nonReentrant modifier in the function described above.

Status

Resolved

8. Missing whenNotPaused modifier, Leads to continue operation even the whole protocol get paused

Path

contracts/HBARProtocol.sol#L211

Function

repayBorrowBehalfNative

Description

repayBorrowBehalfNative function does not have whenNotPaused modifier while repayBorrowNative is so this is the inconsistency in the codebase because of that this function can be called even when the whole protocol is paused which would create problems in the case where because of bug or some other extreme reason it would be necessary to not have any state changes.

Recommendation

We recommend to add whenNotPaused modifier in the repayBorrowBehalfNative function.

Status

Resolved

Informational Issues

9. Unnecessary reentrancy check

Path

contracts/HBARProtocol.sol#L89

Function

supplyUnderlyingNative

Description

In the current implementation, Reentrancy check is un-necessary as no external contract can be called during this function execution. This extra modifier add complexity and consume more gas.

Recommendation

We recommend to remove nonReentrant modifier in the supplyUnderlyingNative function call.

Status

Resolved

10. Missing disableInitializers in constructor leads to initialize the logic contract

Path

contracts/MarketPositionManager.sol

Function

constructor()

Description

MarketPositionManager contract is the upgradeable contract so it has to be initialize through its proxy however it is possible to initialize the logic contract by any one else, To avoid that it is general practice to have disableInitializers() function call in the constructor of the logic contract so no one can initialize the logic contract.

Recommendation

We recommend to call disableInitializers() function in constructor.

Status

Resolved

Closing Summary

Four issues of High severity, two each issue of medium, low & informational severity were found. In the End, Sirio Team Resolved all issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Sirio. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Sirio. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Sirio Team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2024

For

sirio

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com