SYMM IO AUDIT REPORT

January 4, 2024

by

SmartHub Innovations
#SmartAudits
(Ridwan)

CONTENTS

EXECUTIVE SUMMARY

Project Name: SYMM IO

Overview: Reimagining bilateral OTC Derivatives by combining them with Intent-Based execution. Allowing permissionless leverage trading of any asset, with hyperefficient just-in-time liquidity.

Method: Manual Review, Functional Testing, Automated Testing etc.

Scope of Audit: The scope of this audit was to analyze Bank X codebase for quality, security, and correctness.

Codebase: https://github.com/sherlock-audit/2023-12-symm-io-Dar-Sub/tree/main

	High	Medium
Open Issues	2	3
Acknowledged Issues	0	0
Resolved & Closed	0	0

TYPES OF SEVERITIES

High

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 fifixed before moving to a live environment.

Medium

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

Low

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

Informational

These are 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 fifixed.

Acknowledged

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

CHECKED VULNERABILITIES

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

TECHNIQUES AND METHODS

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour
- 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

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

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerability 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 automated analysis were manually verified.

Gas Consumption

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

Tools and Platforms used for Audit

Remix IDE, Hardhat, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

ISSUES FOUND - MANUAL TESTING/CODE REVIEW

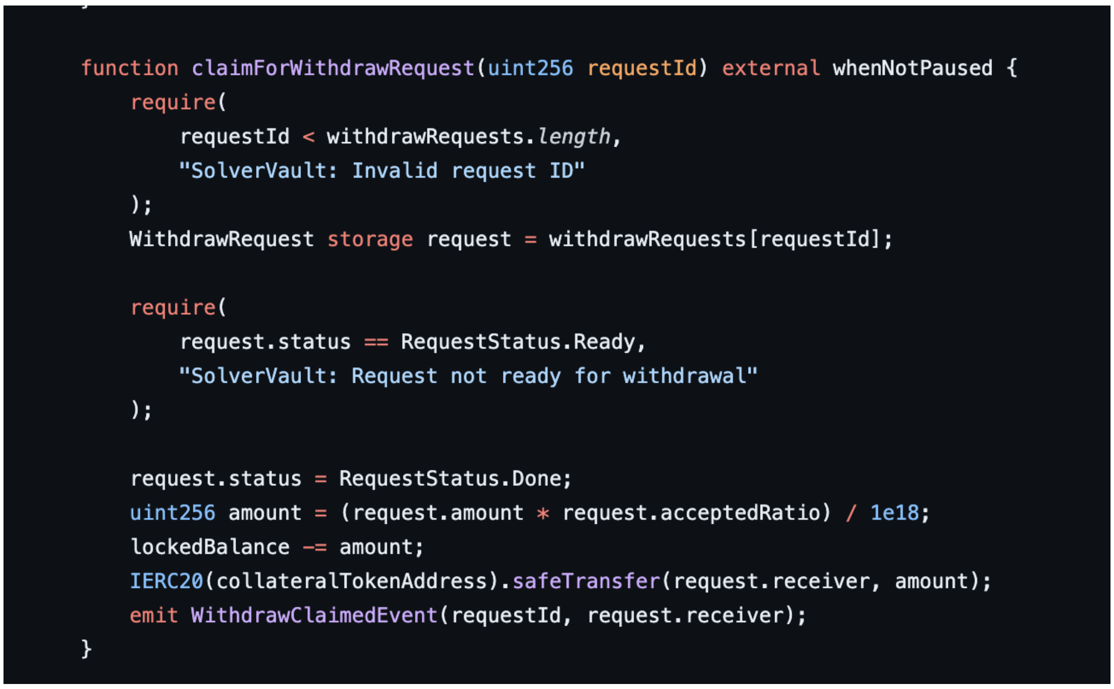
HIGH SEVERITY ISSUES

1. Reentrancy Attack

Contract - SolverVault.sol

Function - `claimForWithdrawRequest`(282 - 299)

Severity; High



```
function claimForWithdrawRequest(uint256 requestId) external whenNotPaused {
    require(
        requestId < withdrawRequests.length,
        "SolverVault: Invalid request ID"
    );
    WithdrawRequest storage request = withdrawRequests[requestId];

    require(
        request.status == RequestStatus.Ready,
        "SolverVault: Request not ready for withdrawal"
    );

    request.status = RequestStatus.Done;
    uint256 amount = (request.amount * request.acceptedRatio) / 1e18;
    lockedBalance -= amount;
    IERC20(collateralTokenAddress).safeTransfer(request.receiver, amount);
    emit WithdrawClaimedEvent(requestId, request.receiver);
}
```

Description & Key Points of concern:

The `claimForWithdrawRequest` function could be susceptible to reentrancy attacks as the state is updated after the external call to safeTransfer. It is generally recommended to update the state before making external calls to avoid reentrancy vulnerabilities.
Risk: An attacker could exploit reentrancy to re-enter the contract and execute malicious code during the state transition.

Remediation

Ensure that state changes are made before making external calls. Use the "checks-effects-interactions" pattern to prevent reentrancy attacks.

2. Missing Access Control Checks

Contract - SolverVault.sol

Function - `acceptWithdrawRequest`(236 - 267)

Severity; High

Description

The acceptWithdrawRequest function lacks access control checks. Any address can call this function, which may lead to unauthorized withdrawals or manipulation of the withdrawal process.
Risk: Unauthorized parties may interfere with the withdrawal process.

Remediation

Add access control checks to restrict the execution of `acceptWithdrawRequest` to authorized roles.

MEDIUM SEVERITY ISSUES

1. Potential Overflow/Underflow

Function - `deposit` & `requestWithdraw`

Severity: Medium

Description:

The contract uses arithmetic operations that might lead to overflow or underflow, such as in the calculation of `amountInSolverVaultTokenDecimals` and `amountInCollateralDecimals`. It's crucial to handle arithmetic operations carefully to prevent unintended behavior.
Risk: Arithmetic overflow or underflow can lead to unexpected behavior or vulnerabilities.

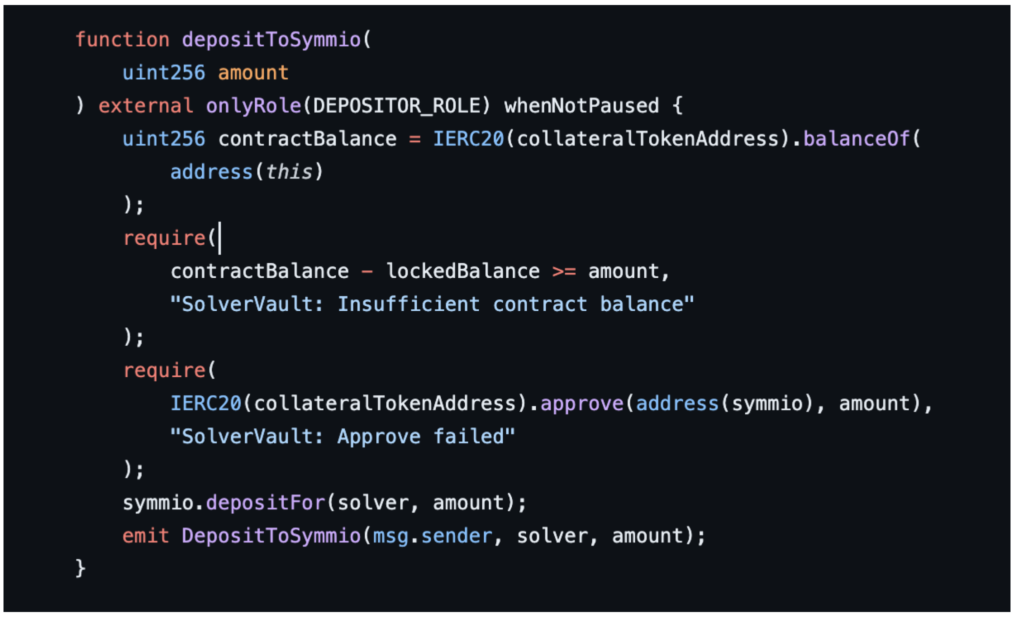
Remediation

Use safe math libraries or checks to prevent overflow/underflow issues.
https://github.com/ConsenSysMesh/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol

2. Approval and Transfer Race Condition

Function - `depositToSymmio` (183 - 199)

Severity: Medium



```
function depositToSymmio(
    uint256 amount
) external onlyRole(DEPOSITOR_ROLE) whenNotPaused {
    uint256 contractBalance = IERC20(collateralTokenAddress).balanceOf(
        address(this)
    );
    require(
        contractBalance - lockedBalance >= amount,
        "SolverVault: Insufficient contract balance"
    );
    require(
        IERC20(collateralTokenAddress).approve(address(symmio), amount),
        "SolverVault: Approve failed"
    );
    symmio.depositFor(solver, amount);
    emit DepositToSymmio(msg.sender, solver, amount);
}
```

Description

In the depositToSymmio function, there is an approval and transfer operation that may result in a race condition. It's possible for the approval to succeed, but the subsequent transfer could fail, leading to unexpected behavior.

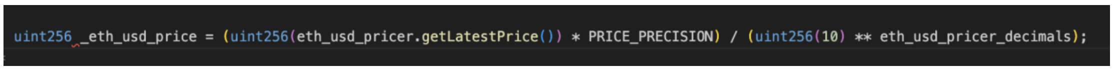
Risk: Race conditions may lead to inconsistencies in the contract state

Remediation

Consider using the `transferFrom` method, which combines the approval and transfer in a single atomic step to avoid race conditions.

3.Lack of Reentrancy Protection in depositToSymmio:

Severity: Medium



```
uint256 _eth_usd_price = (uint256(eth_usd_pricer.getLatestPrice()) * PRICE_PRECISION) / (uint256(10) ** eth_usd_pricer_decimals);
```

Description

The `depositToSymmio` function interacts with an external contract (`symmio.depositFor`) before updating the contract state. This pattern could make the contract susceptible to reentrancy attacks.

Remediation

Apply the "checks-effects-interactions" pattern to ensure that external calls are made after state changes.

AUTOMATED TEST

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the BankX codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the BankX Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

ABOUT SMARTAUDITS

SmartAudits is a secure smart contracts audit platform designed by SmartHub Innovations Technologies.
We are a team of dedicated blockchain security experts and smart contract auditors
determined to ensure that Smart Contract-based Web3 projects can avail the latest and best
security solutions to operate in a trustworthy and risk-free ecosystem.

#SmartAudits