ChainAudit Services

Souq Liquid staking Vault - V1
Security Review Report

Report Version Final

August 8, 2023

Contents

1 Scope of Audit

The scope of this audit was to analyze and document the **Souq NFT Liquid Staking Vault** smart contracts codebase.

2 Introduction

Souq has developed an Automated Market Maker (AMM) for trading non-fungible tokens (NFTs). In the initial phase, stablecoin liquidity pools have been built, allowing users to enter and exit NFT positions quickly and at fair prices. Users have the option to create LP positions by providing liquidity in the form of NFTs or stablecoins to these pools and earn rewards.

Disclaimer : This security review does not guarantee protection against hacking. It represents a snapshot in time of the Souq Liquid staking vault, specific to the commit being referred to. Any modifications made to the code will necessitate a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low
Likelihood: Informational	Low	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix
- Informational - Could Fix

4 Executive Summary

Over the course from June 5 to July 24, Souq Finance Team engaged with ChainAudit Services to review the Liquid Staking Vault smart contract for V1.

In this period of time a total of **18** issues were found and most of them are fixed.

Summary

Project Name	Souq Finance
Repository	Liquid Staking Vault-V1
Commit Hash	bf9326143a539a7676542dee24648f1bb2a07451 4e24e46078cfb4ddd364f2a55a7474fe0d01877a
Type of Project	NFT AMM
Audit Timeline	June 5 to July 24

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	16	9	7
Informational	1	1	0
Total	**18**	**0**	**0**

5 Findings

5.1 Critical Risk

5.1.1 Unchecked Transfer Return Value

Context:
- LPToken.sol#104
- Liquidity1155Logic.sol#147
- Liquidity1155Logic.sol#193
- Liquidity1155Logic.sol#324
- Liquidity1155Logic.sol#450
- Liquidity1155Logic.sol#679
- Liquidity1155Logic.sol#765
- Pool1155Logic.sol#636
- Pool1155Logic.sol#642

Description: The **transfer** and **transferFrom** functions are not being checked. It is essential to validate these return values to ensure the success of the transfer operations. Failing to do so may lead to potential issues, including failed transactions and loss of funds. :

Recommendation: It is recommended to update the contract to use SafeERC20.sol methods.

Status: Fixed #4e24e46

5.2 Low Risk

5.2.1 Unprotected Direct Updates to implementation contract

Context:
- PoolFactory1155.sol#20-154
- AddressesRegistry.sol#31-34

Description: In the OpenZeppelin contracts, an uninitialized contract can be taken over by an attacker. To prevent the implementation contract from being used, the constructor can invoke the ***_disableInitializers*** function to automatically lock it when it is deployed.

This has no direct effect on the proxy storage but still a malicious user may update the state of implementation directly and can attempt to feed other users false information about the protocol by showing the implementation contract

Recommendation: Add _disableInitializers call in implementation contracts.

Status: Acknowledged

5.2.2 Check Divide before multiply

Context: At multiple places in Pool1155Logic.sol#180-199

Description: Solidity's integer division truncates the result. Therefore, performing division before multiplication can result in precision loss. To avoid this issue, it is essential to consider the order of operations and use alternative approaches, such as multiplying before dividing, or utilizing higher precision data types or libraries for precise calculations when handling division and multiplication in smart contracts.

Recommendation: To avoid precision loss in Solidity, it's essential to order multiplication before division in mathematical expressions. By performing multiplication first, it preserves the higher precision of the intermediate result before dividing, reducing the risk of losing precision.

Status: CLosed

5.2.3 Tautology Statement while setting the fee discount

Context: AccessNFT.sol#118

Description: The expression `require(discount >= 0, "Discount must not be less than 0")` is always true and consumes extra gas at the time of setFeeDiscount.

Recommendation: Remove require check to save gas and remove tautology from the system.

Status: Fixed #4e24e46

5.2.4 Ignore return values

Context:
- MME1155.addLiquidityStable ignores return value by Liquidity1155Logic.addLiquidityStable
- MME1155.addLiquidityShares ignores return value by Liquidity1155Logic.addLiquidityShares.
- MME1155.removeLiquidityStable ignores return value by Liquidity1155Logic.removeLiquidityStable.
- MME1155.removeLiquidityShares ignores return value by Liquidity1155Logic.removeLiquidityShares
- MME1155.processWithdrawals(uint256) ignores return value by Liquidity1155Logic.processWithdrawals.
- Pool1155Logic.withdrawFromStableYield ignores return value by aTokenAddress.approve.

Description: When making an external call to another contract in Solidity, it's essential to capture and handle the return value appropriately. Failing to do so by not storing the return value in a local or state variable could lead to unintended consequences and make the result of the external call inaccessible or unused.

Recommendation: Ensure that all the return values of the function calls are used.

Status: Partial Fixed #4e24e46

5.2.5 Missing Zero check Validation

Context:
1. LPToken.constructor._pool
2. AMMBase.constructor._registry
3. MME1155.constructor._factory
4. PoolFactory1155.initialize._poolLogic
5. PoolFactory1155.upgradePools.newLogic
6. AccessNFT.constructor._addressesRegistry

Description: Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities.

Recommendation: We recommend adding a zero-check for the passed-in address value to prevent unexpected errors.

Status: Fixed #4e24e46

5.2.6 Upgradable contracts don’t use gap in between

Context: MME1155 inheriting from AMMBase

Description: Storage gaps are a convention used in upgradable contracts to reserve storage slots in a base contract, allowing future versions of that contract to utilize those slots without affecting the storage layout of child contracts.

To create a storage gap, developers can declare a fixed-size array in the base contract with an initial number of slots. Typically, an array of uint256 is used, where each element reserves a 32-byte slot in storage. The array should be named with a specific pattern, such as __gap or a name starting with __gap_. By using these specific names, OpenZeppelin Upgrades can recognize the storage gap and facilitate smooth upgrades.

Recommendation: Use gap storage and make the layout of MME1155 as per below.

CURRENT LAYOUT

report | graph (this) | graph | inheritance | parse | flatten | funcSigs

Name	Type	Slot	Offset	Bytes	Contract
_initialized	uint8	0	0	1	contracts/amm/MME1155.sol:MME1155
_initializing	bool	0	1	1	contracts/amm/MME1155.sol:MME1155
yieldReserve	uint256	1	0	32	contracts/amm/MME1155.sol:MME1155
poolData	struct DataTypes.PoolData	2	0	864	contracts/amm/MME1155.sol:MME1155
_status	uint256	29	0	32	contracts/amm/MME1155.sol:MME1155
__gap	uint256[49]	30	0	1568	contracts/amm/MME1155.sol:MME1155
__gap	uint256[50]	79	0	1600	contracts/amm/MME1155.sol:MME1155
_paused	bool	129	0	1	contracts/amm/MME1155.sol:MME1155
__gap	uint256[49]	130	0	1568	contracts/amm/MME1155.sol:MME1155
_owner	address	179	0	20	contracts/amm/MME1155.sol:MME1155
__gap	uint256[49]	180	0	1568	contracts/amm/MME1155.sol:MME1155
subPools	struct DataTypes.AMMSubPool1155[]	229	0	32	contracts/amm/MME1155.sol:MME1155
tokenDistribution	mapping(uint256 => uint256)	230	0	32	contracts/amm/MME1155.sol:MME1155
queuedWithdrawals	struct DataTypes.Queued1155Withdrawals	231	0	96	contracts/amm/MME1155.sol:MME1155

RECOMMENDED LAYOUT

report | graph (this) | graph | inheritance | parse | flatten | funcSigs

Name	Type	Slot	Offset	Bytes	Contract
_initialized	uint8	0	0	1	contracts/amm/MME1155.sol:MME1155
_initializing	bool	0	1	1	contracts/amm/MME1155.sol:MME1155
yieldReserve	uint256	1	0	32	contracts/amm/MME1155.sol:MME1155
poolData	struct DataTypes.PoolData	2	0	864	contracts/amm/MME1155.sol:MME1155
__gap	uint256[48]	29	0	1536	contracts/amm/MME1155.sol:MME1155
_status	uint256	77	0	32	contracts/amm/MME1155.sol:MME1155
__gap	uint256[49]	78	0	1568	contracts/amm/MME1155.sol:MME1155
__gap	uint256[50]	127	0	1600	contracts/amm/MME1155.sol:MME1155
_paused	bool	177	0	1	contracts/amm/MME1155.sol:MME1155
__gap	uint256[49]	178	0	1568	contracts/amm/MME1155.sol:MME1155
_owner	address	227	0	20	contracts/amm/MME1155.sol:MME1155
__gap	uint256[49]	228	0	1568	contracts/amm/MME1155.sol:MME1155
subPools	struct DataTypes.AMMSubPool1155[]	277	0	32	contracts/amm/MME1155.sol:MME1155
tokenDistribution	mapping(uint256 => uint256)	278	0	32	contracts/amm/MME1155.sol:MME1155
queuedWithdrawals	struct DataTypes.Queued1155Withdrawals	279	0	96	contracts/amm/MME1155.sol:MME1155

Status: Acknowledged

5.2.7 External calls inside a loop

```
0: LPToken.constructor(address,address,address[],string,string,uint8) (contracts/amm/LPToken.sol#25-39) has external calls inside a l
oop: IERC1155(tokens[i]).setApprovalForAll(address(pool),true) (contracts/amm/LPToken.sol#37)
1: Liquidity1155Logic.depositInitial(address,uint256,uint256,DataTypes.Shares1155Params,DataTypes.PoolData,DataTypes.AMMSubPool1155[]
,mapping(uint256 => uint256)) (contracts/libraries/Liquidity1155Logic.sol#129-153) has external calls inside a loop: IERC1155(poolDat
a.tokens[0]).safeTransferFrom(user,poolData.poolLPToken,params.tokenIds[i],params.amounts[i],) (contracts/libraries/Liquidity1155Logi
c.sol#143)
2: Liquidity1155Logic.addLiquidityShares(address,uint256,DataTypes.Shares1155Params,DataTypes.PoolData,DataTypes.AMMSubPool1155[],map
ping(uint256 => uint256)) (contracts/libraries/Liquidity1155Logic.sol#208-281) has external calls inside a loop: IERC1155(poolData.to
kens[0]).safeTransferFrom(user,poolData.poolLPToken,vars.currentShare.tokenId,vars.currentShare.amount,) (contracts/libraries/Liquidi
ty1155Logic.sol#265-271)
3: Liquidity1155Logic.removeLiquidityShares(address,uint256,DataTypes.Shares1155Params,DataTypes.PoolData,DataTypes.AMMSubPool1155[],
DataTypes.Queued1155Withdrawals,mapping(uint256 => uint256)) (contracts/libraries/Liquidity1155Logic.sol#349-428) has external calls
inside a loop: IERC1155(poolData.tokens[0]).safeTransferFrom(poolData.poolLPToken,user,vars.currentShare.tokenId,vars.currentShare.am
ount,) (contracts/libraries/Liquidity1155Logic.sol#401-407)
4: Liquidity1155Logic.processWithdrawals(uint256,DataTypes.PoolData,DataTypes.Queued1155Withdrawals) (contracts/libraries/Liquidity11
55Logic.sol#438-469) has external calls inside a loop: ILPToken(poolData.poolLPToken).setApproval20(poolData.stable,current.amount) (
contracts/libraries/Liquidity1155Logic.sol#449)
5: Liquidity1155Logic.processWithdrawals(uint256,DataTypes.PoolData,DataTypes.Queued1155Withdrawals) (contracts/libraries/Liquidity11
55Logic.sol#438-469) has external calls inside a loop: IERC20(poolData.stable).transferFrom(poolData.poolLPToken,current.to,current.a
mount) (contracts/libraries/Liquidity1155Logic.sol#450)
6: Liquidity1155Logic.processWithdrawals(uint256,DataTypes.PoolData,DataTypes.Queued1155Withdrawals) (contracts/libraries/Liquidity11
55Logic.sol#438-469) has external calls inside a loop: IERC1155(poolData.tokens[0]).safeTransferFrom(poolData.poolLPToken,current.to,
current.shares[j].tokenId,current.shares[j].amount,) (contracts/libraries/Liquidity1155Logic.sol#453-459)
7: Liquidity1155Logic.swapShares(address,uint256,uint256,DataTypes.Shares1155Params,DataTypes.PoolData,DataTypes.AMMSubPool1155[],map
ping(uint256 => uint256)) (contracts/libraries/Liquidity1155Logic.sol#605-680) has external calls inside a loop: IERC1155(poolData.to
kens[0]).safeTransferFrom(user,poolData.poolLPToken,vars.currentShare.tokenId,vars.currentShare.amount,) (contracts/libraries/Liquidi
ty1155Logic.sol#661-667)
8: Liquidity1155Logic.swapStable(address,uint256,DataTypes.Shares1155Params,DataTypes.PoolData,DataTypes.AMMSubPool1155[],mapping(uin
t256 => uint256)) (contracts/libraries/Liquidity1155Logic.sol#690-768) has external calls inside a loop: IERC1155(poolData.tokens[0])
.safeTransferFrom(poolData.poolLPToken,user,vars.currentShare.tokenId,vars.currentShare.amount,) (contracts/libraries/Liquidity1155Lo
gic.sol#753-759)
```

Description: Using external calls inside loops in Solidity can be risky and lead to higher gas costs. It's generally not recommended to perform external calls inside loops because of the following reasons:

1. **Gas Costs:** Each external call consumes a considerable amount of gas. When called within a loop, this gas cost can quickly add up, leading to high transaction fees and potentially running into gas limits.

2. **Block Gas Limit:** Ethereum has a block gas limit, which restricts the maximum amount of gas that a block can consume. If an external call inside a loop exceeds this limit, the transaction will fail, and any changes made during the loop will be reverted.

3. **Reentrancy Vulnerability:** External calls can introduce reentrancy vulnerabilities if not handled carefully. If the called contract performs another external call back into the original contract, it can lead to unexpected behavior and potential security issues.

Recommendation: To mitigate these risks, consider the following approaches:

1. Minimize External Calls: If possible, redesign the contract logic to minimize external calls inside loops. Sometimes, you can batch operations and make fewer calls to achieve the same result.

2. Use Internal Calls: If the called contract is under your control, consider using internal function calls instead of external calls. Internal calls are more efficient and consume less gas compared to external calls.

3. Limit Loop Iterations: Ensure that the loop does not iterate excessively, and its execution cost stays within reasonable limits.

4. Be Cautious with Reentrancy: If you need to use external calls inside loops, be extremely cautious about reentrancy issues and implement proper checks to prevent them.

Status: Acknowledged

5.2.8 Inherited OwnableUpgradeable uses single step ownership transfer

Context: All functions where ownable is called

Description: During the code review, It has been noticed that Liquid staking contracts use single-step ownership

transfer on the OwnableUpgradeable contract**.**

Recommendation: Consider using Ownable2StepUpgradable contract in the implementation.

Status: Acknowledged

5.2.9 Switch to a new comptable version of solidity

Context: Full repository

Description: Currently the whole system is built on solc 0.8.10 version. Which is not the updated version of solidity.

Recommendation: We recommend switching to a stable new stable version of solc 0.8.20 which is supported by Hardhat. Upgrading may require some adjustments to adapt to the changes introduced in the newer version, but it will ensure that your smart contracts are built on the most recent and secure foundations.

Status: Acknowledged

5.2.10 Reentrancies that allow manipulation of the order or value of events

Context:
- Pool1155Logic.RescueTokens
- Liquidity1155Logic.addLiquidityShares
- Liquidity1155Logic.addLiquidityStable
- Pool1155Logic.deployLPToken
- PoolFactory1155.deployPool
- Liquidity1155Logic.depositInitial
- Pool1155Logic.depositIntoStableYield
- MME1155.pause
- Liquidity1155Logic.processWithdrawals
- Liquidity1155Logic.removeLiquidityShares
- Liquidity1155Logic.removeLiquidityStable
- Liquidity1155Logic.swapShares
- Liquidity1155Logic.swapStable
- MME1155.unpause
- AddressesRegistry.updateImplementation
- AddressesRegistry.updateProxy
- PoolFactory1155.upgradePools
- Pool1155Logic.withdrawFees
- Pool1155Logic.withdrawFees
- Pool1155Logic.withdrawFromStableYield

Description: Detects reentrancies that allow manipulation of the order or value of events.

Recommendation: Apply the check-effects-interactions pattern always.

Status: Fixed **#4e24e46**

5.2.11 Usage of block.timestamp

Context:
- Liquidity1155Logic.processWithdrawals
- AccessNFT.HasAcces

Description: Absolutely, using **block.timestamp** for security-critical comparisons in smart contracts can be dangerous, as it can be manipulated by miners to some extent. The **block.timestamp** represents the current block's timestamp in seconds since the Unix epoch, and miners have some control over this value.

Miners have the ability to manipulate the timestamp within certain bounds (typically a few seconds) to some extent. This means that if the smart contract relies on **block.timestamp** for important decisions, malicious miners could potentially set the timestamp to their advantage, leading to unpredictable behavior or even security vulnerabilities.

To avoid these issues, consider using Oracle services to obtain reliable timestamps from off-chain sources for any time-dependent operations. This ensures that your smart contract relies on trustworthy and tamper-resistant time information or applies a 15-second rule which says if the scale of the time dependent event can vary by 15 seconds and maintain integrity.

#SWC-116

Recommendation: Use oracle service or 15-second rule**.**

Status: Acknowledged

5.2.12 Boolean constants can be used directly and don't need to be compared.

Context:
- Liquidity1155Logic.sol#239
- Liquidity1155Logic.sol#373
- Liquidity1155Logic.sol#561
- Liquidity1155Logic.sol#628
- Liquidity1155Logic.sol#709
- AccessNFT.sol#52

Description: Detects the comparison to boolean constants.

Recommendation: Remove the equality to the boolean constant.

Status: Fixed **#4e24e46**

5.2.13 Not used locked pragma version

Context: contracts/interfaces/IERC20.sol#4

Description: The pragma versions used in the interface are not locked. Consider using the compatible versions 0.8.10 for deploying the contracts and libraries as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

```
pragma solidity ^0.8.0; // bad: compiles between 0.8.0 and 0.8.10
pragma solidity 0.8.0; // good : compiles w 0.8.0 only but not the latest version
pragma solidity 0.8.10; // best: compiles w 0.8.10
```

Recommendation: use the best compile version

Status: Fixed **#4e24e46**

5.2.14 Pool1155Logic has cyclomatic complexity

Context: Pool1155Logic.CalculateShares

Description: CalculateShares has high cyclomatic complexity to the system

Recommendation: We recommend splitting the function into several smaller subroutines.

Status: Acknowledged

5.2.15 Removal of unused function and state variables

Context:
- MathHelpers.convertFromWadSqrd
- LPToken._underlyingAsset

Description: convertFromWadSqrd function and _underlyingAsset state variable is not used anywhere and can be removed.

Recommendation: We recommend removing the unused function and state variable.

Status: Fixed #4e24e46

5.2.16 The contract reads its own variable using this keyword

Context: AccessNFT.sol#55

Description: reads this.balanceOf with an extra STATICCALL.

Recommendation: We recommend reading the variable directly from storage instead of calling the contract with this keyword.

Status: Fixed #4e24e46

5.3 Informational

5.3.1 Gas optimizations

Context: Full contract

Description:

- The pre-increment operation is cheaper (about 5 GAS per iteration) so use ++i instead of i++ or i+= 1 in for loop. I recommend using pre-increment in all the for loops.
- In for loop, the default value initialization to 0 should be removed from all the for loops.
- In the EVM, there is no opcode for non-strict inequalities (>=, <=) and two operations are performed (> + =.) Consider replacing **>=** with the strict counterpart **>**. Recommend following the inequality with a strict one.

  The compiler uses opcodes **GT** and **ISZERO** for solidity code that uses **>**, but only requires **LT** for **>=**, [which saves **3 gas**] #REF**.**

Recommendation: We recommend optimisation at various places in the contracts.

Status: Fixed #4e24e46