SALUS SECURITY

JULY 2023

CODE SECURITY ASSESSMENT

FURION

Overview

Project Summary

- Name: Furion - SeparatePool
- Version: commit 77857a3
- Platform: EVM-compatible chains
- Language: Solidity
- Repository: https://github.com/Furion-Finance/Furion
- Audit Range: See Appendix - 1
- Address: 0x2ef7F474f4460188535E84f2d7A7082899a52897

Project Dashboard

Application Summary

Name	Furion - SeparatePool
Version	v3
Type	Solidity
Dates	July 28 2023
Logs	July 13 2023; July 18 2023; July 28 2023

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	3
Total Low-Severity issues	0
Total informational issues	8
Total	13

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	Locked NFTs can be bought by others	High	Business Logic	Resolved
2	Downcasting of NFT id from uint256 to uint16 can lead to collision	High	Business Logic	Resolved
3	Unbounded loop for allNfts can lead to denial of service	Medium	Denial of Service	Resolved
4	Incorrect fee variable used in the lockBatch function	Medium	Business Logic	Resolved
5	Incorrect implementation of the getAllPools function	Medium	Denial of Service	Resolved
6	Use of floating pragma	Informational	Configuration	Acknowledged
7	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledged
8	Lack of indexed event parameters	Informational	Logging	Acknowledged
9	Pool creation may revert without proper error messages	Informational	Logging	Acknowledged
10	Missing zero address check	Informational	Data Validation	Acknowledged
11	Inconsistent comment and error message	Informational	Inconsistency	Resolved
12	Spelling mistakes	Informational	Code Quality	Resolved
13	Gas optimization suggestions	Informational	Gas Optimization	Resolved

2.2 Notable Findings

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

1. Locked NFTs can be bought by others	
Severity: High	Category: Business Logic
Target: - contracts/separate-pool/SeparatePool.sol	

Description

The function `lock()` and `lockBatch()` is used to lock the NFTs. Once the NFT is locked, only the locker of the NFT is able to claim it before the releaseTime. If the NFT is not claimed by the locker before the releaseTime, it is open for the public to buy.

But in the current code, the users who are not the lockers can still buy the NFTs before the releaseTime of locked NFT which is an undesired situation. Due to this, the locker of NFT may not be able to claim the NFT before the releaseTime if it has already been bought by other users.

Recommendation

Consider adding a check to ensure that the NFT which the user is buying is not the locked NFT with a releaseTime more than the current timestamp.

Status

This issue has been resolved by the team.

2. Downcasting of NFT id from uint256 to uint16 can lead to collision	
Severity: High	Category: Business Logic
Target: - contracts/separate-pool/SeparatePool.sol	

Description

The inPool array has the type of uint16 which is used to store the id of NFTs in the pool. The `_transferInNFT` function downcasts the NFT id from uint256 to uint16. This unsafe downcasting can lead to collision between the ids of two NFTs. The max value of uint16 is 65535. So, NFT of id 4 and 65540 will be stored as 4 in the inPool array. This can also cause problems in the `_removeElement` function when trying to delete the latter element, it will delete the former because it starts the loop from 0. The totalSupply of kitties is more than type(uint16).max and since different NFTs are integrated, there is a high possibility of this issue.

Recommendation

It is recommended to change the type of the inPool array from uint16 to uint256 and avoid downcasting the NFT id.

Status

This issue has been resolved by the team. The inPool array has been removed. The SeparatePool contract only supports NFTs that implement the ERC721 standard now.

3. Unbounded loop for allNfts can lead to denial of service	
Severity: Medium	Category: Denial of Service
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

Every time a pool is created, an _nftAddress is pushed into the allNfts array. Over time, the size of the allNfts array will grow huge. Iterating over the array may lead to a large amount of gas which may be more than the block gas limit resulting in denial of service when calling the `getAllPools`, `getNftByPool` and `transferOwnership` functions.

Recommendation

It is recommended to avoid all the actions executed in a single transaction. Consider implementing pagination logic to retrieve or search for pools, and splitting the ownership transfer into several calls in case of facing gas limit.

Status

This issue has been resolved by the team. The team has removed the getAllPools and getNftByPool functions. The owner of a SeparatePool can be updated individually via SeparatePool.changeOwner().

4. Incorrect fee variable used in the lockBatch function	
Severity: Medium	Category: Business Logic
Target: - contracts/separate-pool/SeparatePool.sol	

Description

In the `lockBatch` function, the buyFee variable is used to calculate the total fee instead of the lockFee variable.

contracts/separate-pool/SeparatePool.sol:L215-L234

```
function lockBatch(uint256[] calldata _ids) external {
    ...
    uint256 mintTotal = LOCK_MINT_AMOUNT * length;
    uint256 feeTotal = buyFee * length;
    FUR.transferFrom(msg.sender, incomeMaker, feeTotal);
    ...
}
```

Recommendation

Consider changing `buyFee` to `lockFee`.

Status

This issue has been resolved by the team.

5. Incorrect implementation of the getAllPools function	
Severity: Medium	Category: Denial of Service
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

The `getAllPools` function is used to get all the pools for NFTs in the factory contract.

For loop is used for iterating over the NFTs and getting the corresponding pool address. While iterating, it is missing the increment of `i` which will lead to `i` remaining as it is. Thus, the loop will never stop and consumes all gas sent with the transaction.

In memory, dynamically-sized arrays should be allocated with the exact size before use. The unallocated memory array poolAddresses in the `getAllPools` function could lead to a panic during execution.

contracts/separate-pool/SeparatePoolFactory.sol:L60-L71

```
function getAllPools()
        external
        view
        returns (address[] memory poolAddresses)
{
        for (uint256 i; i < allNfts.length; ) {
            address nftAddress = allNfts[i];
            poolAddresses[i] = getPool[nftAddress];
        }

        return poolAddresses;
}
```

Recommendation

Consider adding the increment operator in the loop so that loop iterates over all NFTs, and initializing the poolAddresses variable with the statement `poolAddresses = new address[](allNfts.length);`.

Status

This issue has been resolved by the team.

2.3 Informational Findings

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

Description

```
pragma solidity ^0.8.0;
```

The SeparatePool contracts use a floating compiler version ^0.8.0.

Using a floating pragma ^0.8.0 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.

7. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

The SeparatePoolFactory 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.

8. Lack of indexed event parameters	
Severity: Informational	Category: Logging
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

Indexed parameters are useful for quick off-chain filtering. During the new pool deployment, none of the parameters in the PoolCreated event are indexed.

contracts/separate-pool/SeparatePoolFactory.sol:L23-L27

```
event PoolCreated(
       address nftAddress,
       address poolAddress,
       uint256 poolIndex
);
```

Recommendation

Consider indexing applicable event parameters to support the searching and filtering abilities of off-chain services.

Status

This issue has been acknowledged by the team.

9. Pool creation may revert without proper error messages	
Severity: Informational	Category: Logging
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

During the pool creation, the _tokenMetadata function is called, which will get the NFT name and symbol for token metadata. However, it is optional to implement IERC721Metadata according to the ERC721 standard, which means the pool creation is not available for some NFTs and the transaction may be reverted.

contracts/separate-pool/SeparatePoolFactory.sol:L157-L158

```
string memory nftName = IERC721Metadata(_nftAddress).name();
string memory nftSymbol = IERC721Metadata(_nftAddress).symbol();
```

Recommendation

It is recommended to add a try/catch statement to handle this special case and throw proper error messages.

Status

This issue has been acknowledged by the team.

10. Missing zero address check	
Severity: Informational	Category: Data Validation
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

contracts/separate-pool/SeparatePoolFactory.sol:L29-L37

```
constructor(
    address _incomeMaker,
    address _checker,
    address _fur
) {
    incomeMaker = _incomeMaker;
    Checker = IChecker(_checker);
    fur = _fur;
}
```

It is considered a security best practice to verify addresses against the zero address in the constructor.

contracts/separate-pool/SeparatePoolFactory.sol:L114-L117

```
require(
    address(Checker) != address(0),
    "SeparatePoolFactory: Checker not set."
);
```

Since there is no way to update the Checker after deployment, the above code in the createPool function can be moved to the constructor.

Recommendation

Consider making changes based on the above suggestions.

Status

This issue has been acknowledged by the team.

11. Inconsistent comment and error message

Severity: Informational	Category: Inconsistency
Target: - contracts/separate-pool/SeparatePool.sol	

Description

The `lockBatch` function is used to lock the NFTs in batches. However, the comment and the error message use the term "buy", which is inconsistent.

contracts/separate-pool/SeparatePool.sol:L215-L218

```
function lockBatch(uint256[] calldata _ids) external {
    // Number of NFTs to buy
    uint256 length = _ids.length;
    require(length < 10, "SeparatePool: Can only buy 9 NFTs at once");
    ...
}
```

Recommendation

Consider replacing the term "buy" with "lock" in the comment and error message.

Status

This issue has been resolved by the team.

12. Spelling mistakes	
Severity: Informational	Category: Code Quality
Target: - contracts/separate-pool/SeparatePoolFactory.sol	

Description

contracts/separate-pool/SeparatePoolFactory.sol:L130

```
// New way to invoke create2 without assembly, paranthesis still needed for empty
constructor
```

The word "paranthesis" is misspelled, and should be "parenthesis" instead.

Recommendation

Consider correcting the spelling mistakes.

Status

This issue has been resolved by the team.

13. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - contracts/separate-pool/SeparatePoolFactory.sol - contracts/separate-pool/SeparatePool.sol	

Description

contracts/separate-pool/SeparatePoolFactory.sol:L20-L21

```
IChecker Checker;
address public fur;
```

contracts/separate-pool/SeparatePool.sol:L17

```
ERC20 FUR;
```

To reduce gas costs, the above state variables could be declared as immutable since their values are fixed after the contracts have been deployed.

contracts/separate-pool/SeparatePoolFactory.sol:L60-L71

```
function getAllPools()
    external
    view
    returns (address[] memory poolAddresses)
{
    for (uint256 i; i < allNfts.length; ) {
        address nftAddress = allNfts[i];
        poolAddresses[i] = getPool[nftAddress];
    }

    return poolAddresses;
}
```

In the `getAllPools()`, `getNftByPool()`, and `transferOwnership()` functions, direct usage of `allNfts.length` is observed when iterating through allNfts, as mentioned in the code snippet above. It is recommended to use a temporary variable to cache the length to save gas.

Recommendation

Consider making changes based on the above suggestions.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 77857a3:

File	SHA-1 hash
contracts/separate-pool/SeparatePool.sol	9cc2c648b8cbeb62ba10aefb035a8e5044823e6d
contracts/separate-pool/SeparatePoolFactory.sol	be19bd9e15f331900d45d1797c2fa477aeb31ff4
contracts/separate-pool/interfaces/ISeparatePool.sol	5c8997c4a60c0e6f3972fa7f6fb915319b108849
contracts/separate-pool/interfaces/ISeparatePoolFactory.sol	5ad5285a9e2b60744411771dbf5629cc838f6c24