Audit Report for Scattering

Dec 31, 2023

Overview

Scattering protocol aims to make NFTs more liquid to increase accessibility. Users can break their NFTs into ERC20–based fragment tokens. By trading fragment tokens like other ERC20 tokens, NFT owners gain much more flexibility in managing their NFT assets. On the opposite, users can also redeem the fragment tokens into NFTs, which further improves the liquidity for NFTs.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on–chain monitoring are strongly recommended.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact – the technical, economic and reputation damage of a successful attack

Likelihood – the chance that a particular vulnerability gets discovered and exploited

Severity – the overall criticality of the risk

Apart from the issues, this audit report also includes recommendations for developers. For those potential issues that will not be fixed on the code level but mitigated by offchain methods, this report will mark them as **Notes.**

Security Assessment Summary

Scope

The audit is conducted based on Scattering protocol github repository with commit hash *c90f178d6b87a1b73d18d4dc459fe2446648b9f6*. **All issues are fixed** at commit *4aa0a6fc12338c5f9c96caee8c1f22f5b560b99c*. The audit scope includes all Solidity files in the repository.

The following number of issues were found, categorized by their severity:

- Critical & High: 0
- Medium: 1
- Low: 1
- Recommendations: 0
- Notes: 1

Findings Summary

ID	Title	Severity
[M-01]	Dangerous virtual function	Low
[L-01]	Lack of sanity check	Low
[N-01]	Mitigated sandwich risks	–

Detailed Findings

[M-01] Dangerous virtual function

Severity

Impact: High, the dangerous virtual function can lead to arbitrary calls.

Likelihood: Low, current implementations all override the dangerous virtual function

Status: Fixed

Description

The `Scattering` and `ScatteringPeriphery` contracts inherit `Multicall` contract. The `Multicall` contract has a dangerous virual function `extMulticall` .

```
1
2       function extMulticall(CallData[] calldata calls) external
    virtual override returns (bytes[] memory) {
3           return multicall2(calls);
4       }
5   function multicall2(CallData[] calldata calls) internal returns
    (bytes[] memory) {
6           bytes[] memory results = new bytes[](calls.length);
7           CallData calldata calli;
8           for (uint256 i = 0; i < calls.length; ) {
9               calli = calls[i];
10              (bool success, bytes memory result) =
    calli.target.call(calli.callData);
11              if (success) {
12                  results[i] = result;
13              } else {
14                  // Next 4 lines from
15                  // https://github.com/OpenZeppelin/openzeppelin-
    contracts-
    upgradeable/blob/master/contracts/utils/AddressUpgradeable.sol#L2
    29
16                  if (result.length > 0) {
17                      assembly {
18                          let returndata_size := mload(result)
19                          revert(add(32, result), returndata_size)
20                      }
21                  } else {
```

```
                    revert FailedMulticall();
                }
            }

            unchecked {
                ++i;
            }
        }
        return results;
    }
```

Although both contracts override the dangerous function by adding a modifier `onlyRole(ADMIN_ROLE)` to restrict its accessibility, there is still a possibility that the developers may mistakenly forget to override it in later development.

Recommendations

Remove the `multicall2` invocation in `extMulticall`.

[L-01] Lack of sanity check

Severity

Impact: Medium, users can bypass the costs to invoke specific functions

Likelihood: Low, the impact can only happen when the contract is misconfigured

Status: Fixed

Description

In `Scattering`, users can keep their NFTs from being auctioned for a limited time to redeem them back in the future. `Scattering` allows users to extend the limited time by paying specific tokens.

```
function extendKeys(
        address collection,
```

```
        uint256[] memory nftIds,
        uint256 newRentalDays
    ) external payable nonReentrant whenNotPaused {
        _mustValidNftIds(nftIds);
        _mustValidRentalDays(newRentalDays);

        (CollectionState storage collectionState, address underlying) = _useUnderlyingCollectionState(collection);
        // verify paymentAmount and nftLengths(_mustValidNftIds already valid) is greater than zero
        if (paymentAmount > 0) {
            uint256 totalPayoutAmount = paymentAmount * nftIds.length;
            _addTokensInternal(address(this), paymentToken, totalPayoutAmount);
        }

        collectionState.extendLockingForKeys(
            //            userFloorAccounts[msg.sender],
            LockParam({
                proxyCollection: collection,
                collection: underlying,
                //                    creditToken: creditToken,
                nftIds: nftIds,
                rentalDays: newRentalDays
            }),
            msg.sender // todo This can be extended here to support renewing keys for other users.
        );
    }
```

However, if the `paymentAmount` is misconfigured to `0`, the users can extend the time for free. As such, the configuration function should be implemented with correct sanity checks.

```
function setPaymentParam(address _paymentToken, uint256 _paymentAmount) external onlyRole(ADMIN_ROLE) {
        paymentToken = _paymentToken;
        paymentAmount = _paymentAmount;
    }
```

Recommendations

Check the `paymentAmount` must not be `0` when setting payment parameters.

[N-01] Mitigated sandwich risks

Description

Users can utilize `ScatteringPeriphery` to interact with `Scattering`. The contract provides functions for users to swap fragment tokens to other tokens via AMM pools.

```
function fragmentAndSell(
        address collection,
        uint256[] calldata tokenIds,
        bool unwrapWETH,
        ISwapRouter.ExactInputParams memory swapParam
    ) external payable nonReentrant returns (uint256 swapOut) {
        (, uint32 commonPoolCommission, ) = commissionInfo();
        // nftLen * FLOOR_TOKEN_AMOUNT * (10000+200)/10000
        uint256 fragmentTokenAmount = tokenIds.length *
            Constants.FLOOR_TOKEN_AMOUNT *
            ((10_000 + commonPoolCommission) / 10_000);

        address fragmentToken = fragmentTokenOf(collection);

        /// approve all
        approveAllERC721(collection, address(_scattering));
        approveAllERC20(fragmentToken, uniswapRouter,
fragmentTokenAmount);

        /// transfer tokens into this
        ERC721Transfer.safeBatchTransferFrom(collection,
msg.sender, address(this), tokenIds);

        /// fragment
        _scattering.fragmentNFTs(collection, tokenIds,
msg.sender);
        IERC20(fragmentToken).transferFrom(msg.sender,
address(this), fragmentTokenAmount);

```

```
26          swapOut =
    ISwapRouter(uniswapRouter).exactInput(swapParam);
27
28          if (unwrapWETH) {
29              unwrapWETH9(swapOut, msg.sender);
30          }
31      }
```

There is a potential risk in the swapping procedure. MEV bots can sandwich this transaction to profit from the user. `Scattering` protocol mitigated this risk by adding a default slippage protection for its users when generating the swap parameters.