code4rena

Fractality

Smart Contract Security Assessment

Version 2.0

Audit dates: Aug 06 — Aug 08, 2024

Audited by: peakbolt
thereksfour
k4zanmalay

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 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

2. Executive Summary

2.1 About Fractality

Fractality is a lending protocol that allows users to earn a yield through different strategies, pro-share basis. The yield is offered to users and in return the protocol uses the liquidity provided to run low risk basis trading strategy on centralized exchanges (CEXs)/decentralized perpetual exchanges, liquidity provision on variety of low risk pairs/vaults & more.

2.2 Scope

Repository	Fracticality-Protocol/FractalityV2Vault
Commit Hash	2a6df5a40c8e9bc55cd5b87bf651db18e00d67c4%C2%A7
Mitigation Hash	218b66e4e94e41500c6702e04e662fed9c19d1f7

2.3 Audit Timeline

DATE	EVENT
Aug 06, 2024	Audit start
Aug 08, 2024	Audit end
Oct 29, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	6
Low Risk	1
Informational	2
Total Issues	9

3. Findings Summary

ID	DESCRIPTION	STATUS
M-1	`requestRedeem()` can be DoS for specific `controller`	Resolved
M-2	Pending withdrawals dilute the exchange rate	Resolved

M-3	Users may experience losses due to slippage issues	Resolved
M-4	Users who requested assets redemptions may be affected by the new `claimableDelay` and `redeemFeeBasisPoints`	Resolved
M-5	Possible underflow in redeem() can prevent users from redeeming their assets	Resolved
M-6	The vault will not work with tokens that do not return values on transfer (e.g. USDT)	Resolved
L-1	`maxDeposit()` and `maxMint()` should return 0 when halted	Resolved
I-1	`Withdraw()` event should use after-fees asset amount	Resolved
I-2	Vault does not support rebasing token	Acknowledged

4. Findings

4.1 Medium Risk

A total of 6 medium risk findings were identified.

[M-1] `requestRedeem()` can be DoS for specific `controller`

Severity: Medium	Status: Resolved

Context:

- FractalityV2Vault.sol#L794-L834

Description: `requestRedeem()` can be requested for a specific `controller` address by anyone, and only one request can be made for each `controller` at any time.

This allows an attacker to request redeem for a victim `controller` address with dust shares and then not call `redeem()` to leave it as pending indefinitely. This will block legitimate redeem request for that `controller` as new redeem request cannot be created till the pending request is redeemed.

```
function requestRedeem(
    uint256 shares,
    address controller,
    address owner
)
    external
    onlyWhenNotHalted
    operatorCheck(owner)
    nonReentrant
    returns (uint8)
{
    if (controller == address(0) || owner == address(0)) {
        revert ZeroAddress();
    }
    uint256 assets = convertToAssets(shares);
    if (assets == 0) {
        revert ZeroAssets();
    }
```

```
            RedeemRequestData storage request =
redeemRequests[controller];

        if (request.redeemRequestCreationTime > 0) {
            revert ExistingRedeemRequest();
        }
```

Recommendation:

This can be fixed by introducing multiple redeem requests for each controller, allowing more than one redeem request for each `controller` address. Each request shall be uniquely identified by both `controller` and `requestId`.

- It is important that each redeem request have a different `requestId`, as ERC-7540 requires request with the same `requestId` to be fungible (i.e. requests with same `requestId` must be claimable at the same exchange rate).
- As per ERC-7540, requests with different `requestId` do not have the fungible requirements and can be claimable at different times and at different exchange rate. Hence, this is recommended.

As discussed, we do not recommend adding to existing redeem request as there is no good solution on updating the `redeemRequestCreationTime` in a fair manner. That is because if we allow subsequent request to update the creation time (e.g to the latest creation time), it will allow one to indefinitely delay the redeem time. Another issue of bypass the redeem delay could also occur if we try to take an average of the creation time, as that will allow part of the redeem request to be claimable at an earlier time.

Fractality: Resolved with the operator functionality

Zenith: Verified

[M-2] Pending withdrawals dilute the exchange rate

Severity: Medium	Status: Resolved

Context:

- FractalityV2Vault.sol#L884-L885

Description:

When users call requestRedeem(), their shares are sent to the contract. And when users call redeem(), these shares are burned. These shares in the contract will dilute the exchange rate, and only when users call redeem() to redeem the pending withdrawals, the exchange rate will be correct.

1. Say total assets are 1000 and total shares are 1000.
2. Alice request redeem for 500 shares.
3. Then reportProfits increases total assets to 2000.
4. The exchange rate will be 2000/1000 = 2, but since the exchange rate for the withdrawn 500 shares is locked, the correct exchange rate should be (2000-500)/(1000-500) = 3.

Recommendation:

It is recommended to reduce vaultAssets and burn shares in requestRedeem().

Fractality: We will fix this by burning shares (and related operations) on redeem request.

Zenith: Resolved

[M-3] Users may experience losses due to slippage issues

Severity: Medium	Status: Resolved

Context:

- FractalityV2Vault.sol#L808

Description: When a redemption request is made, the vault will calculate the amount of assets the user will receive for his shares:

```
    function requestRedeem(
        uint256 shares,
        address controller,
        address owner
    )
        external
        onlyWhenNotHalted
        operatorCheck(owner)
        nonReentrant
        returns (uint8)
    {
        if (controller == address(0) || owner == address(0)) {
            revert ZeroAddress();
        }
>>      uint256 assets = convertToAssets(shares);
        if (assets == 0) {
            revert ZeroAssets();
        }

        RedeemRequestData storage request = redeemRequests[controller];

        if (request.redeemRequestCreationTime > 0) {
            revert ExistingRedeemRequest();
        }

        request.redeemRequestShareAmount = shares;
>>      request.redeemRequestAssetAmount = assets;
```

The problem may occur if there is a significant delay between sending `requestRedeem` transaction and it's execution. For example the share price may go down and user will receive fewer assets than expected.

Consider the following scenario:

- Bob is the only shareholder with 1000 shares, and there are 1000 USDC in the vault
- he calls `requestRedeem` and expects to receive 1000 USDC in exchange for 1000 shares
- his transaction was delayed for some reason (low gas amount, arbitrum sequencer down) and `reportLosses` was executed first, leaving only 500 USDC in the vault
- when Bob's transaction is finally executed, he will have only 500 USDC in the request.

Recommendation: To fully comply with ERC7540 it is recommended to create an overloaded version of the `requestRedeem` function in which user can specify minimum amount of assets he wants to receive:

```
    function requestRedeem(
        uint256 shares,
        address controller,
        address owner
    )
        external
        returns (uint8)
    {
         return _requestRedeem(shares, controller, owner);
    }

    function requestRedeem(
        uint256 shares,
        uint256 minAssetsOut,
        address controller,
        address owner
    )
        external
        returns (uint8)
    {
         if(convertToAssets(shares) < minAssetsOut) revert
MinAmountFail();
         return _requestRedeem(shares, controller, owner);
    }
```

Fractality: Fixed in PR-1

Zenith: Verified

[M-4] Users who requested assets redemptions may be affected by the new `claimableDelay` and `redeemFeeBasisPoints`

Severity: Medium	Status: Resolved

Context:

- FractalityV2Vault.sol#L457-L462

Description: Increasing `claimableDelay` will result in unfair extension of the redemption for users who submitted a request before the change was made.

```
    function setClaimableDelay(
        uint32 _newClaimableDelay
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        claimableDelay = _newClaimableDelay;
        emit ClaimableDelaySet(_newClaimableDelay);
    }
```

And increasing `redeemFeeBasisPoints` will result in unfair charges of the redemption for users who submitted a request before the change was made.

```
    function setRedeemFee(
        uint16 _newRedeemFeeBasisPoints
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        if (_newRedeemFeeBasisPoints > _MAX_BASIS_POINTS) {
            revert InvalidRedeemFee();
        }
        redeemFeeBasisPoints = _newRedeemFeeBasisPoints;
        emit RedeemFeeSet(_newRedeemFeeBasisPoints);
    }
```

Recommendation: Calculate the redemption date and fee at the time of request creation:

```
-       request.redeemRequestCreationTime = uint96(block.timestamp);
+       request.redeemRequestFulfillTime = uint96(block.timestamp) +
claimableDelay;
+       request.redeemRequestFee = _calculateWithdrawFee(assets);
```

claimable check:

```
        if (
-           block.timestamp < request.redeemRequestCreationTime +
claimableDelay
+           block.timestamp < request.redeemRequestFulfillTime
        ) {
```

fee transfer:

```
        uint256 netAssetRedeemAmount = request.redeemRequestAssetAmount -
            request.redeemRequestFee;

        if (
            !asset.transfer(redeemFeeCollector, request.redeemRequestFee)
||
            !asset.transfer(receiver, netAssetRedeemAmount)
        ) {
            revert ERC20TransferFailed();
        }
```

Fractality: Fixed in PR-1

Zenith: The issue with fees is verified, and the delay issue is acknowledged.

[M-5] Possible underflow in redeem() can prevent users from redeeming their assets

Severity: Medium	Status: Resolved

Context:

- FractalityV2Vault.sol#L887

Description:

If the call order is requestRedeem() -> reportLosses() -> redeem(), vaultAssets may be less than request.redeemRequestAssetAmount, resulting in underflow. For example, Alice deposits 10000 tokens, vaultAssets is 10000. Alice call requestRedeem(), and redeemRequestAssetAmount is 10000. Then reportLosses() is called and vaultAssets reduce to 9000. When Alice call redeem(), since redeemRequestAssetAmount > vaultAssets, the subtraction `vaultAssets -= request.redeemRequestAssetAmount` will underflow and revert the transaction.

Recommendation: It is recommended to reduce vaultAssets in requestRedeem(), and also burn shares in requestRedeem().

Fractality: We will fix this by burning shares (and related operations) on redeem request.

Zenith: Verified

[M-6] The vault will not work with tokens that do not return values on transfer (e.g. USDT)

Severity: Medium	Status: Resolved

Context:

- FractalityV2Vault.sol#L901-L902
- FractalityV2Vault.sol#L935
- FractalityV2Vault.sol#L942
- FractalityV2Vault.sol#L975

Description: Some tokens doesn't fully comply with ERC20 standard and do not return boolean values on `transfer` and `transferFrom` calls, one of these tokens is USDT. Since the vault contract checks for a successful transfer and expects to receive a return value from the asset, USDT cannot be used as an underlying token.

One example of such a transfer call:

```
        if (!asset.transferFrom(msg.sender, strategy.strategyAddress,
assets)) {
            revert ERC20TransferFailed();
        }
```

Recommendation: Consider using SafeTransferLib from Solmate to transfer assets.

Fractality: Fixed in PR-1

Zenith: Verified

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] `maxDeposit()` and `maxMint()` should return 0 when halted

Severity: Low	Status: Resolved

Context:

- FractalityV2Vault.sol#L592-L609

Description:

`maxDeposit()` and `maxMint()` should return 0 when halted, similiar to `maxRedeem()`. This is aligned with ERC4626 specification where both functions are required to return 0 when deposits/mints are disabled.

```
    function maxDeposit(
        address /*receiver*/
    ) public view override returns (uint256) {
        uint256 remainingCapacity = maxVaultCapacity - vaultAssets;
        return
            remainingCapacity < maxDepositPerTransaction
                ? remainingCapacity
                : maxDepositPerTransaction;
    }

    function maxMint(address receiver) public view override returns
(uint256) {
        return convertToShares(maxDeposit(receiver));
    }
```

Recommendation:

Update `maxDeposit()` and `maxMint()` to return 0 when halted.

```
    function maxDeposit(
        address /*receiver*/
    ) public view override returns (uint256) {

+        if (halted) {
+            return 0; //cannot deposit when halted
```

```
+         }

          uint256 remainingCapacity = maxVaultCapacity - vaultAssets;
          return
              remainingCapacity < maxDepositPerTransaction
                  ? remainingCapacity
                  : maxDepositPerTransaction;
      }

      function maxMint(address receiver) public view override returns
  (uint256) {
          return convertToShares(maxDeposit(receiver));
      }
```

Fractality: Fixed as recommended, under commit https://github.com/Fracticality-Protocol/FractalityV2Vault/commit/db3775a8133dabbb9771567edb9501cd45aafea4

Zenith: Verified

4.3 Informational

A total of 2 informational findings were identified.

[I-1] `Withdraw()` event should use after-fees asset amount

Severity: Informational	Status: Resolved

Context:

- FractalityV2Vault.sol#L907-L913

Description:

According to ERC-4626, `previewRedeem()` and `previewWithdraw()` must be inclusive of fees so that integrators are aware of the withdrawal fees. This implies that `Withdraw` event should be based on actual asset amount received by the receiver, after accounting for fees. This is also explained in the OZ doc for [Custom behavior: Adding fees to the vault]. (https://docs.openzeppelin.com/contracts/5.x/erc4626#fees)

However, the protocol emits `Withdraw()` event based on `request.redeemRequestAssetAmount`, which is not inclusive of fees.

```
        emit Withdraw(
            msg.sender,
            receiver,
            request.originalSharesOwner,
            request.redeemRequestAssetAmount,
            shares
        );
```

Recommendation:

Consider using `netAssetRedeemAmount` instead of `request.redeemRequestAssetAmount` as follows:

```
        emit Withdraw(
            msg.sender,
            receiver,
            request.originalSharesOwner,
-           request.redeemRequestAssetAmount,
+           netAssetRedeemAmount
            shares
```

```
        );
```

Fractality: Fixed as recommended, under commit https://github.com/Fracticality-Protocol/FractalityV2Vault/commit/db3775a8133dabbb9771567edb9501cd45aafea4

Zenith: Verified

[I-2] Vault does not support rebasing token

Severity: Informational	Status: Acknowledged

Context:

- FractalityV2Vault.sol#L583-L585

Description:

The vault asset accounting is synthetic. `totalAssets()` uses `vaultAssets` that is updated based on deposit/mint and not based on `balanceOf()`. That means the vault will not be able to support rebasing tokens.

```
    function totalAssets() public view override returns (uint256) {
        return vaultAssets;
     }
```

Recommendation: Consider acknowledging this and add documentation to only use vault asset that are non-rebasing token or wrapper for re-basing token.

Fractality: Documented to make sure a rebasing token is not used as the asset in commit https://github.com/Fracticality-Protocol/FractalityV2Vault/commit/db3775a8133dabbb9771567edb9501cd45aafea4

Zenith: Verified